UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ana Cláudia Fiorin Pianesso

Identificação e Classificação de Comportamentos de Objetos Dinâmicos

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Orientação
Raul Sidnei Wazlawick

Florianópolis, abril de 2002.

Identificação e Classificação de Comportamentos de Objetos Dinâmicos

Ana Cláudia Fiorin Pianesso

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Conhecimento e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Fernando Alvaro Ostuni Gauthier, D. Sc.
Coordenador do Curso

Banca Examinadora

Prof. Raul Sidnei Wazlawick, D. Sc.
Orientador

Profa. Edla Maria Faust Ramos, D. Sc.

Prof. Vitorio Bruno Mazzola, D. Sc.

À minha mãe por ter acreditado que eu chegaria ao fim e por ter depositado em mim seus sonhos, vibrando calada a cada conquista minha como se fosse sua própria vitória.

AGRADECIMENTOS

Agradeço:

À minha família, em especial à minha mãe, *Ser* indispensável nesta conquista merecedora de todo meu reconhecimento. E aos meus irmãos Andréia, Luciano e Aluísio, pela paciência e compreensão quando da minha ausência e abstinência.

Ao meu noivo Antônio Carlos, pelo apoio, carinho e incentivo em todos os momentos fazendo-me a pessoa mais capaz do mundo.

Ao Professor Raul Sidnei Wazlawick, pela orientação e disponibilidade ao longo do trabalho.

Aos amigos e colegas, pela convivência diária que fez de cada um uma personalidade marcante e inesquecível. Em especial à Josiane, Priscila, Luis Claudio, Sandro e Tati.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O presente trabalho tem por objetivo a criação de uma biblioteca de padrões de comportamentos de objetos dinâmicos, considerando os comportamentos já implementados em aplicações utilizando a ferramenta Mundo dos Atores. Com os padrões de comportamentos encontrados nestas aplicações pode-se elaborar uma primeira abordagem de uma biblioteca conceitual de padrões de comportamentos predefinidos, para que a ferramenta Mundo dos Atores possa ser adaptada para o trabalho com desenvolvedores de aplicações que não possuem conhecimento específico em programação de computadores. Oferecendo condições à prática da criatividade e capacidade de projetar ou manipular elementos de *software,* sem precisar necessariamente entender como foram feitos ou como funcionam. Atendo-se desta forma nos conceitos pertencentes ao tema em estudo, e não em conceitos de criação de representação dos conceitos utilizados pela ferramenta.

Para a realização deste trabalho faz-se uso de conceitos de padrões de projeto (*design patterns*), pois aproveitando-se de idéias que se repetem e que já foram testadas e apresentaram-se satisfatórias, os padrões de projetos podem acelerar o processo de criação de artefatos de software, podendo ser aplicados no contexto de programação de comportamentos de objetos dinâmicos. O trabalho tem como enfoque a ferramenta Mundo dos Atores, visto esta ter se mostrado adequada no processo de ensino de programação para alunos do curso de ciência da computação. Da mesma forma, esta ferramenta apresentou-se satisfatória na concepção e prototipação de software educacional produzido por professores de primeiro e segundo graus praticamente leigos em computação.

ABSTRACT

The present work aims the setting up of behaviours in dynamic objects, considering behaviours already implemented in the applications using the "Mundo dos Atores" *tool. Firstly, when behaviours patterns are found into those applications, a first approach for the conceptual library of predefined behaviours patterns may be elaborated enabling such tool to be customised for the application users who are not into to computer programming. This library will supply the application users, conditions to practice their creativity and skills to create and handle software elements, without having the need to understand how they were made or functions. Therefore, the application users will aim the subject of study rather than concepts that have to be used in this tool representation.*

Design patterns concepts were used, taking advantage of pretested and successfully repeated ideas used previously, which speed up the creation process of software devices, such pattern approach may be inserted into the dynamic object behaviours programming context. The "Mundo dos Atores" *tool has been used in this work because it fits into the programming teaching process for students of the computer science course. Therefore this tool had a satisfactory result in the conception and prototyping of a educational software done by primary and secondary school teachers who have mainly basic computer skills.*

1 Introdução

A programação pode ser uma atividade divertida e potenciadora, muitas vezes acessível a quem consegue ultrapassar a grande barreira inicial. Esta barreira é constituída entre outros aspectos, pela aprendizagem de uma linguagem de programação formal e conceitos computacionais como variáveis, procedimentos, controle de fluxo, classes e objetos (Kahn, 1995) gerando desta forma, uma sobrecarga cognitiva em iniciantes ou leigos em computação. Se fosse possível minimizar esta barreira e ultrapassar o que restar com divertimento e criatividade, as crianças e os adultos sem conhecimento específico em computação, seriam capazes de moldar os programas à forma que desejassem, não necessariamente condicionados a aprendizagem de uma linguagem de programação.

Para minimizar o esforço e diminuir o tempo necessário para produzir artefatos de software, pode-se recorrer à Engenharia de Software, que dispõe de diferentes abordagens que buscam alcançar estes aspectos, objetivando a qualidade dos mesmos (Fairley, 1986). A reutilização de software, neste contexto, é um fator que pode levar aos aspectos citados anteriormente. A reutilização de componentes de software existentes em novos sistemas implica numa menor produção de software novo, causando um aumento da produtividade e redução do tempo de desenvolvimento, devido à diminuição do esforço necessário para a produção de códigos novos. O aumento da qualidade advém do fato do código reutilizado já ter sido amplamente usado, modificado e testado em outros sistemas eliminando boa parte dos erros que poderiam ocorrer (Furlan, 1995) acelerando e simplificando o processo de desenvolvimento (Silva, 2000).

Para que ocorra reutilização de software, um dos requisitos é a produção de artefatos reutilizáveis. Os princípios que norteiam a modularidade, como a criação de módulos com pequenas interfaces, explícitas e em pequena quantidade, e o uso do princípio da ocultação de informação (Coplien & Schmidt, 1995), tem sido o principal elemento de orientação para a produção de bibliotecas de artefatos reutilizáveis (Silva, 2000). Esta reutilização pode abranger o nível de código (consistindo na utilização direta de trechos de código já desenvolvidos), considerada a forma mais comum de reutilização, sendo que a criação de bibliotecas é utilizada para sistematizar esta prática. Outra forma é a reutilização de projeto (reaproveitamento de concepções arquitetônicas de um artefato em outro artefato de software).

Uma das formas de minimizar a carga de trabalho, ou mesmo a carga cognitiva envolvida na produção de software por leigos ou não em computação, é aproveitar-se de artefatos reutilizáveis reunidos em uma biblioteca (Coplien & Schmidt, 1995), consistindo na utilização direta de trechos de código já desenvolvidos. Assim, o desenvolvedor não precisa preocupar-se com a implementação dos artefatos contidos nesta biblioteca, podendo-se valer ainda de características de padrões de projeto, pois parte-se do pressuposto que esta é resultado de experiências anteriores que deram certo e se repetem para um determinado domínio de problema.

O aumento de produtividade considerando-se a prototipação de software ou a própria criação de aplicações, está diretamente relacionado com a granularidade do artefato reutilizado. Assim, o desenvolvedor pode-se aproveitar da reutilização de módulos e não necessariamente criá-los. Esta reutilização pode ocorrer sob dois enfoques diferentes (Silva, 2000): na forma de composição - uso de classes disponíveis em uma biblioteca para originar os objetos da implementação; na forma de herança - aproveitamento da concepção de classes disponíveis em uma biblioteca, no desenvolvimento de outras.

Neste trabalho uma biblioteca é considerada uma coleção de artefatos de software reunidos, sendo estes artefatos comportamentos de objetos dinâmicos. Onde, estes comportamentos são trechos de códigos representando o conjunto de

ações simples condicionadas que devem ser executadas em cada instante de tempo, combinados com o princípio da ocultação de informação.

Este trabalho juntamente com os conceitos apresentados anteriormente procura abordar questões relacionadas ao projeto *MuseuVirtual*. Dentre elas, a proposta deste projeto de propiciar condições para que os usuários da ferramenta resultante da pesquisa possam atribuir comportamentos aos objetos criados, através de uma biblioteca de comportamentos predeterminados. Assim, os usuários não necessitam de um conhecimento específico de programação para elaborar os comportamentos dos objetos da sua aplicação. A descrição do projeto *MuseuVirtual* é apresentada na próxima seção.

Vale ressaltar que um problema no contexto deste trabalho, é o fato de saber se o conjunto de comportamentos contidos na biblioteca proposta aqui é suficiente para programar outros comportamentos. Isso se consegue fazendo uma prova de equivalência da linguagem gerada com a *Máquina de Turing* ou outro mecanismo equivalente, não estando no escopo do trabalho.

1.1 Contexto do Trabalho – *O Projeto Museu Virtual*

MuseuVirtual é um projeto de pesquisa que propõe fornecer uma ferramenta de autoria distribuída multi-usuário, na qual estudantes (ensino médio inicialmente), em locais diferentes possam simultaneamente construir objetos em ambientes de realidade virtual definindo comportamentos para estes objetos.

A motivação para esta pesquisa foi a observação de que os ambientes virtuais disponíveis na Internet normalmente possuem várias limitações em termos de interatividade entre usuários (Wazlawick, Kirner et al, 2001). A ferramenta *MuseuVirtual* adota conceitos de museus virtuais em conformidade com uma nova abordagem de museus que se opõem à visão de que um museu virtual é apenas um site para manter cultura (Yamada, Hong, & Sugita, 1995) mas que deve abranger as expectativas dos visitantes com características de interatividade. Para tanto, o *MuseuVirtual* suporta vários tipos de interação nos mundos virtuais que venham ser

construídos.

Dentre as diferentes formas de interatividade pesquisadas, a ferramenta de autoria permite aos professores e estudantes criarem seus próprios ambientes virtuais, onde objetos dinâmicos interativos e *chatterbots*[1] podem ser programados. Pode-se, ainda, projetar objetos necessários para representar um determinado micro-mundo em uma ferramenta gráfica, e estes serem importados para a ferramenta *MuseuVirtual*, programando-se então comportamentos específicos para estes objetos.

A aprendizagem das diferentes matérias curriculares e não curriculares não é limitada pela simples visitação[2] a um ambiente existente, e sim, apoiada por projetos nos quais os alunos são engajados na construção de seus próprios ambientes virtuais não imersivos[3]. Isto pode ocorrer pela proposição por parte do professor e/ou alunos a depender do tema em estudo.

O aluno pode conduzir as pesquisas do tema em questão, projetar ambientes virtuais individualmente, ou colaborativamente, representando os conceitos envolvidos no estudo. Estes ambientes, por sua vez, podem ser visitados por outros alunos ou grupos interessados no tema. Desta forma, o ambiente virtual construído pelo aluno, será resultante do seu processo de aprendizagem (Wazlawick et al, 2001).

Esta proposta é fortemente baseada em conceitos de programação por atores e, do ponto de vista pedagógico adota conceitos da abordagem construtivista (Piaget, 1967) (Nova Escola, 1995). Acredita-se que esta abordagem aplica e melhora os modelos cognitivos e cenários pedagógicos, promovendo um poderoso aprendizado interativo e uma efetiva colaboração autônoma entre estudantes (Wazlawick, Kirner et al, 2001).

Muitos autores tem identificado a autoria em realidade virtual promissora

---

[1] Agente que ouve e responde ao usuário, servindo como guia virtual do ambiente.
[2] Entende-se visitação como a navegação e interação do usuário com o mundo virtual. Na visitação o usuário neste caso pode ser representado por um *avatar*.
[3] Ambientes não imersivos dispensam o uso de dispositivos especiais para navegação, como luvas e capacetes.

para fins educacionais. Entretanto, devido a complexidade envolvida na autoria de mundos virtuais, muitas aplicações consistem apenas em mundos pré definidos, onde estudantes podem conversar, e em muitos casos interagir, mas não podem criar seus próprios ambientes. Assim, a autoria em realidade virtual nas escolas é um desafio (Wazlawick, Kirner et al, 2001). A questão é, como fornecer ferramentas de autoria com poder suficiente, permitindo por exemplo, a programação de comportamentos de objetos 3D reativos e proativos, e que possam ser entendidos e usados por não-especialistas no assunto.

A autoria de comportamentos de objetos é dificultada quando os usuários têm pouco ou até nenhum conhecimento específico na linguagem de programação a ser usada. O uso de comandos, sintaxe de programação, sintaxe de erros, são questões que oneram o aprendizado destes usuários. Deve-se, então, fornecer uma alternativa que permita o melhor entendimento de programação de comportamentos (Wazlawick, Kirner et al, 2001).

Para que, os usuários da ferramenta possam praticar sua criatividade e serem capazes de projetar ou manipular elementos de *software* como por exemplo, inserir objetos dinâmicos reativos e pró-ativos, definindo o comportamentos dos objetos (série de ações simples condicionadas que devem ser executadas em cada instante de tempo), ou mesmo criação dos próprios comportamentos sem treinamento específico em computação ou em uma linguagem de programação, faz-se necessário o uso de uma ferramenta intuitiva e adequada.

A ferramenta Mundo dos Atores (Mariani, 1998) tem se mostrado adequada no processo de ensino de programação para alunos do curso de ciência da computação. Da mesma forma, a ferramenta demonstrou ser satisfatória na concepção e prototipação de *software* educacional produzido por professores do ensino fundamental e médio sem conhecimento em computação. Entretanto, várias dificuldades foram encontradas na aplicação desta ferramenta. Entre elas a necessidade do uso de uma linguagem de programação, que embora muito simples e flexível, produz uma sobrecarga cognitiva não desejável no processo (Wazlawick et al, 2001).

Uma das formas de adequar a ferramenta Mundo dos Atores para o trabalho com crianças e adolescentes com pouco, ou até mesmo nenhum conhecimento específico de programação, é disponibilizar um conjunto preestabelecido de comportamentos para os objetos. Desta forma, pode-se manipular os comportamentos dos objetos, sem precisar necessariamente entender, como foram feitos ou como funcionam. Esta abordagem pode reduzir significativamente a necessidade de ensino de conceitos de programação, visto que, não é este o objetivo do processo, além de simplificar a criação de aplicações. Por conseguinte, reduz-se o tempo de dedicação, e o processo de desenvolvimento de aplicações, ou dos objetos e seus comportamentos transcorrerá com menor complexidade.

Uma abordagem que pode auxiliar no desenvolvimento do conjunto preestabelecido de comportamentos é a análise dos projetos já existentes na busca de padrões de projeto. Com o aproveitamento de idéias que se repetem em diferentes aplicações, que já foram testadas apresentando-se satisfatórias pode-se acelerar o processo de criação de artefatos de *software*.

1.2 Objetivos

1.2.1 Objetivo Geral

O principal objetivo deste trabalho consiste na criação de uma biblioteca de padrões de comportamentos predefinidos de objetos, para que o Mundo dos Atores possa ser adaptado para o trabalho com desenvolvedores de aplicações sem conhecimento específico em programação de computadores.

Como exemplo disto, pode-se citar que, na versão atual do Mundo dos Atores, se o usuário desejar construir um ator capaz de detectar a presença de outro ator, ou o toque, ele deverá escrever linhas de código que efetuem este comportamento a partir de uma linguagem de programação (*smalltalk*). Com a criação de padrões de comportamento, o usuário poderá simplesmente indicar que ele quer que o ator detecte a presença de outro ator, realizando uma determinada ação em resposta a esta presença. Não se pretende, no entanto, a substituição total de linhas de códigos na programação de comportamentos dos objetos e sim, sugerir

a atribuição de comportamentos através de uma interface gráfica seguindo a tendência de um estilo de programação visual.

1.2.2 Objetivos Específicos

Como objetivos específicos deste trabalho podem ser citados:

a) Reconhecimento dos diferentes padrões de comportamentos nos diversos sistemas já implementados utilizando o Mundo dos Atores. A partir destes padrões, tem-se um ponto de partida para a construção de outros comportamentos que possam oferecer aos alunos um conjunto de comportamentos de mais alto nível, que atenda as suas necessidades.

b) Criação de uma biblioteca conceitual com comportamentos mais gerais, baseando-se nos comportamentos encontrados na fase de análise dos projetos já implementados. Dentre estes, propor comportamentos básicos para a criação de outros. Com isso os comportamentos existentes em projetos anteriores, podem ser alcançados através de composição de comportamentos mais gerais. Possibilitando assim, a criação de N combinações de comportamentos, não descartando a possibilidade de criação de comportamentos diferentes dos já existentes.

1.3 Hipóteses e Limitações do Trabalho

A biblioteca de comportamentos resultante tem que oferecer recursos para a criação de um número significativo de comportamentos de objetos. Os comportamentos encontrados nos projetos já implementados devem ser obtidos através de combinação de novos comportamentos disponibilizados na biblioteca. Assim como, possibilidade de criação de novos comportamentos complexos a partir dos comportamentos básicos e de construir uma classificação sistemática para eles (atendendo a expectativa do usuário).

Uma das limitações deste trabalho é a verificação do conjunto de

comportamentos para saber se são suficientes para implementar as aplicações já existentes, ou seja, os comportamentos já programados pelos usuários. Esta verificação pode ser obtida através de uma prova de equivalência da linguagem gerada com a *Máquina de Turing* ou outro mecanismo equivalente. Esta prova de equivalência não está no escopo do trabalho pela limitação do tempo disponível para elaboração do mesmo.

1.4 Resultados Esperados

Com os comportamentos constituintes da biblioteca, espera-se a simplificação do processo de criação de aplicações com a ferramenta "Mundo dos Atores". Com a disponibilização de comportamentos predefinidos para objetos e possibilidade de criação de novos comportamentos através dos já existentes, o tempo de dedicação no processo de desenvolvimento de aplicações é reduzido. Da mesma forma, a carga cognitiva envolvida deixa de ser um fator preocupante com influência direta no processo de criação. Isso deve-se ao fato de que os comportamentos constituintes da biblioteca deverão ser bastante intuitivos, possibilitando a manipulação dos comportamentos dos objetos, sem a necessidade de entendimento de como foram programados.

1.5 Estrutura da Dissertação

O próximo capítulo (capítulo 2) aborda conceitos de padrão de projeto. São apresentadas as origens dos padrões, justificativas para seu uso, maneiras de identificá-los, seus tipos e componentes.

Vislumbra-se no capítulo 3, o panorama atual da utilização da ferramenta "Mundo dos Atores". Além da breve apresentação da ferramenta, é abordada a experiência de sua utilização com alunos.

No capítulo 4, por sua vez, é apresentada a análise da experiência de utilização da ferramenta "Mundo dos Atores". São descritos os critérios utilizados para análise dos projetos dos alunos objetivando a identificação de padrões de comportamentos programados para objetos dinâmicos.

O capítulo 5, apresenta a construção de um modelo conceitual dos padrões de comportamentos encontrados na fase de análise dos projetos dos alunos. São abordados conceitos relacionados a utilização dos comportamentos constituintes da biblioteca proposta, bem como exemplos da aplicação dos comportamentos.

No capítulo 6 são apresentadas as considerações finais do trabalho e sugestões para continuidade do mesmo.

2 Padrões de Projeto

Uma estrutura de projeto bem feita tem um impacto positivo no desenvolvimento de software. Um programador familiarizado com estas estruturas pode aplicá-las em problemas sem ter que redescobri-las. Estruturas de projetos (*design*) também facilitam o reuso de arquiteturas bem sucedidas, melhoram a documentação e manutenção de sistemas existentes, fornecendo especificações explícitas de classes e interações de objetos e suas intenções, ou seja, o que eles realmente querem fazer (Ambler, 1998).

Padrões constituem uma abordagem recente em termos de reutilização de projeto, no contexto do desenvolvimento de software orientado a objetos (Silva, 2000). A questão é: como reutilizar em um desenvolvimento de software a experiência de projeto adquirida em desenvolvimentos anteriores.

2.1 Origens dos Padrões de Projeto

O arquiteto Chistopher Alexander estudou formas para melhorar o processo de construção de edifícios e áreas urbanas (Gamma et al, 1994). Para ele, a natureza de cada lugar é determinada pelos padrões de eventos humanos que acontecem regularmente neste lugar, estando interligados com padrões geométricos do espaço. Estes padrões de eventos no espaço são os únicos elementos que se repetem em diferentes lugares. A combinação de instâncias destes padrões formam fragmentos de conhecimentos para a construção de lugares.

Considerando estes aspectos, ele desenvolveu um padrão de linguagem (*pattern language*) para permitir às pessoas projetarem suas próprias casas e comunidades. Com isso, conseguiu chegar aos chamados padrões de projeto (*design patterns)*, que seriam descrições literárias (texto estruturado) dos padrões encontrados, e o define da seguinte forma:

"Cada padrão de design é uma regra de três que expressa uma relação entre um contexto, um problema e uma solução" (Alexander et al, 1977).

Pode-se dizer, de forma sucinta, que *"um design pattern é uma solução para um problema em um determinado contexto*" (Gamma et al, 1994), onde:

a) **Contexto** se refere ao conjunto de situações que se repetem, nas quais o *design pattern* pode ser aplicado;

b) **Problema** se refere ao conjunto de "forças" – objetivos e limitações – que ocorrem dentro do contexto;

c) **Solução** é uma estrutura formal para ser aplicada na resolução do problema. Tenta capturar a essência do problema, a fim de que outros desenvolvedores a entendam, e possam fazer uso dela em situações similares.

Esta definição pode ser escrita de uma outra maneira:

"É uma relação ternária entre um determinado contexto, um determinado conjunto de forças (objetivos e restrições) que ocorrem repetidamente neste contexto, e uma determinada configuração de software que permite que estas forças sejam resolvidas" (Coplien & Schmidt, 1995).

Padrões de projeto podem ser considerados como uma pura reutilização de idéias que já foram aplicadas no passado e deram certo. Ajudando desta forma gerir a complexidade do desenvolvimento de software, descrevendo elementos de granularidade superior ao objeto e à classe, fornecendo um esquema predefinido para implementação.

Em termos de orientação a objetos, os padrões de projetos identificam

classes, instâncias, seus papéis, colaborações e a distribuição de responsabilidades. Seriam, então, descrições de classes e objetos que se comunicam, que são implementados a fim de solucionar um problema comum em um contexto específico (Ambler, 1999). Em outras palavras, eles correspondem a uma micro-arquitetura em que são definidas as classes envolvidas, suas responsabilidades e a forma como cooperam (Silva, 2000). Desta forma padrões de projeto podem ser considerados como técnicas para a construção de sistemas orientados a objetos, visando ao máximo sua reusabilidade, explicando porque uma determinada solução é necessária e não simplesmente identificando-a.

2.2 Por que usar Padrões de Projeto?

Padrões de projeto têm várias características semelhantes às classes utilizadas no processo de desenvolvimento de software orientado a objetos. Em orientação a objetos, tanto as classes como os padrões de projeto têm dois aspectos básicos (Gamma et al, 1994):

a) **Externo, a visão do problema:** descrições das propriedades, responsabilidades, capacidades e serviços prestados ao software ou ao ambiente externo;

b) **Interno, a visão da solução:** descrições estáticas e dinâmicas, limitações, e relacionamentos entre os componentes, colaboradores, cada um apenas com uma visão externa incompleta.

Padrões de projeto aumentam a expressividade e o nível de descrição suportados pela orientação a objetos. Além disso, pode-se citar outras características dos padrões de projeto, como:

a) Formam um vocabulário comum permitindo uma melhor comunicação entre os desenvolvedores, uma documentação mais completa e uma melhor exploração das alternativas de projeto;

b) Reduzem a complexidade do sistema através da definição de abstrações que estão acima das classes e instâncias. Aumentando desta forma a qualidade do programa;

c) Constituem uma base de experiências reutilizáveis para a construção de software. Funcionam como peças na construção de projetos de software mais complexos. Podem ser considerados como micro-arquiteturas que contribuem para arquitetura geral do sistema;

d) Reduzem o tempo de aprendizado de uma determinada biblioteca de classes. Isto é fundamental para o aprendizado dos desenvolvedores novatos;

e) Quanto mais cedo são usados, menor será o trabalho em etapas mais avançadas do projeto.

2.3 Detectando um Padrão de Projeto

Nem toda solução, algoritmo ou prática constitui um padrão. Embora possa parecer que possua todos os requisitos básicos para ser um padrão, ele não pode ser considerado como tal até que o fenômeno da repetição seja verificado (Gamma et al, 1994). No entanto, deve-se usar padrões de projeto em soluções que se repetem com variações. Não faz sentido o reuso, se o problema só aparece em um determinado contexto. Em geral, padrões de projeto representam soluções que requerem vários passos. Se o número de passos for pequeno, não há a necessidade de usar padrões.

Padrões de projeto se aplicam muito bem em situações em que o desenvolvedor está mais interessado na existência de uma solução do que na sua total implementação. Isto ocorre, na maior parte das vezes, quando o domínio do problema é o foco da questão. Para isso, um padrão de projeto deve aplicar soluções adequadas que um novato não pensaria de imediato; não deve estar dependente especificamente de um tipo de sistema, linguagem de programação ou

domínio de aplicação, sendo genéricos para um problema específico; bem elaborado; simples e reusável para sistemas orientados a objetos.

Segundo James Coplien et al (1996), um bom padrão tem as seguintes características:

a) Resolve o problema, e não apenas o identifica ou indica princípios e estratégias abstratas, capturando soluções;

b) É um conceito provado, não constituindo apenas uma hipótese ou uma especulação;

c) A solução não é óbvia, geram soluções para um problema de forma indireta, ao contrário de várias técnicas de resolução de problemas que tentam derivar soluções a partir de primeiros princípios;

d) Não apenas descreve os módulos, mas a maneira como eles se relacionam, através de estruturas e mecanismos do sistema;

e) Atinge a qualidade quer seja na clareza, na utilidade, na documentação, servindo desta forma para o conforto humano.

2.4 Tipos de Padrões

A expressão *padrão de projeto* muitas vezes se refere a qualquer padrão relacionado a arquitetura, projeto e implementação de software. Buschmann et al (1996), definem 3 tipos de padrões:

a) **Padrão de Arquitetura (*Architectural Pattern*):** representa um esquema de organização estrutural dos sistemas. Fornece um conjunto de subsistemas predefinidos, especificando suas responsabilidades, e incluindo regras para organizar os relacionamentos entre eles. Preocupam-se com o sistema como um todo, seus macro-componentes e suas propriedades globais. Suas implicações afetam a estrutura e a organização do sistema

inteiro.

b) **Padrão de Projeto (*Design Pattern*):** provê um esquema de refinamento dos subsistemas e componentes do software, bem como os relacionamentos entre eles. Descrevem situações que se repetem, e soluções para problemas em contextos específicos. Não afetam a estrutura global do sistema.

c) **Gíria (*Idiom*):** é um padrão de baixo nível, específico a uma linguagem de programação. Uma gíria implementa aspectos dos componentes e de suas relações entre si, utilizando uma determinada linguagem de programação.

2.5 Componentes de Um Padrão de Projeto

Existem diversas formas de se escrever um padrão de projeto (Coplien, 1994). No entanto, independente de formato, alguns componentes devem ser facilmente reconhecidos quando da leitura de um padrão de projeto. A forma aqui descrita é conhecida como forma canônica.

a) **Nome e Classificação** - Deve expressar a essência do padrão. Um bom nome é vital, pois vai se tornar parte do vocabulário do projeto.

b) **Propósito** - O que faz o padrão de projeto? Qual o problema que se propõe a atacar? Qual os objetivos que deseja alcançar?

c) **Motivação** - Descreve o cenário no qual se aplica, todas as forças que estão presentes e as classes e os objetos relacionados.

d) **Aplicabilidade** - Quais são as situações nas quais o padrão de projeto pode ser aplicado? Como reconhecer estas situações?

e) **Participantes** - Descreve as classes e/ou objetos que participam no padrão de projeto e suas responsabilidades.

f) **Colaborações** - Descreve como os participantes colaboram entre si para realizar suas responsabilidades. Assim, a solução descreve não somente a

estrutura estática, mas também a dinâmica comportamental dos objetos e classes.

g) **Diagrama** - Representação gráfica do padrão utilizando a técnica de modelagem de objetos.

h) **Conseqüências** - Quais os resultados obtidos com a aplicação do padrão? O que foi resolvido, o que não foi resolvido e que padrões de projeto podem ser aplicados neste momento?

i) **Implementação** - Quais são as dicas, os "macetes" e as técnicas que devem estar claras quando da implementação do padrão? Há questões relativas a uma linguagem específica?

j) **Exemplos** - Exemplos de sistemas reais, onde o padrão foi aplicado, e que transformações ocorreram dentro de cada contexto.

k) **Padrões de projeto afins** - Quais padrões de projeto têm relação com este? Quais são as diferenças importantes? Com que outros padrões este deve ser usado?

Este formato pode ser simplificado, pois é apenas uma das formas para se escrever um padrão de projeto. Não sendo necessários todos os elementos citados anteriormente na sua descrição. O formato de padrões de projeto identificados neste trabalho não compreenderá todos estes elementos, mas apenas aqueles julgados mais relevantes no momento da análise

3 Mundo dos Atores

A ferramenta "*Mundo dos Atores*" (Mariani, 1998), inspirada na linguagem *Logo* (Papert, 1980), foi criada para suportar a aprendizagem pelo método de ensino não seqüencial. Para interagir com a ferramenta é dispensável o conhecimento prévio sobre programação de computadores, mas necessário um certo conhecimento referente a operações com um ambiente computacional. Assim, o *Mundo dos Atores* (www.inf.ufsc.br/~mariani) é indicado para utilização em disciplinas introdutórias de programação de computadores (Mariani, 2001), podendo também ser utilizado para explorar conceitos matemáticos, ou mesmo exercitar aspectos como a criatividade.

O Mundo dos Atores é composto de um palco representado por uma área gráfica como ilustrado na figura 1, e de um espaço para comandos (área inferior). O aprendiz então, trabalha com a "metáfora de palco". Fazendo-se uma analogia com um teatro, o lugar onde os atores atuam é o *Palco*. São apresentados elementos virtuais como a caneta, os atores e o palco, tendo estes correspondência direta com elementos concretos equivalentes. Fazendo com que o aprendiz manipule *objetos,* antes mesmo que o conceito seja formalmente apresentado.

O palco é o ambiente onde os atores (objetos) existem e interagem. Cada ator mantém uma referência para o palco, e este para os atores nele incluídos. Cada palco e cada ator possui um conjunto de ações básicas que podem ser conhecidas pelo aprendiz. Além destas, novas ações podem ser incorporadas ao repertório do palco ou dos atores, e o palco também pode ser modificado para se adequar a uma nova situação. Novos palcos podem ser criados por especialização dos existentes, visto a ferramenta ser aberta

e expansível (Mariani, 1998).

Da mesma forma que o palco, os atores também podem ser especializados. Cada ator pode estar associado a uma espécie de roteiro que indica qual seu papel no palco. A definição de papéis possibilita o estabelecimento de comportamentos diferenciados para os atores, em particular no que se refere à movimentação no palco e à interação com outros atores.

Iniciado o processo de animação, cada ator passa a "desempenhar" seu papel de forma independente. O comportamento geral do sistema resulta dos comportamentos individuais de cada ator (Mariani, 1998). O controle da execução dos papéis dos atores é realizado pelo palco através do mecanismo de paralelismo por eventos discretos. Cada ator então, a cada instante, executa sem interrupções cada sentença incondicional e cada sentença ou ações definidas, as quais uma condição associada é verdadeira (Wazlawick et al, 2001).

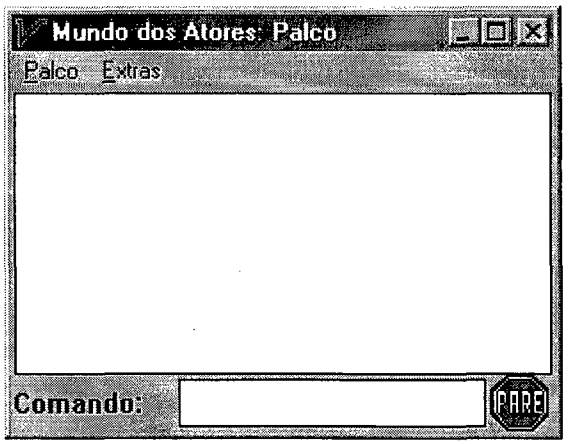

Figura 1: Interface Gráfica do Mundo dos Atores

A primeira versão da ferramenta utilizava uma metáfora similar à de *Logo*, pela existência de gráficos e um objeto semelhante à tartaruga, controlável através de comandos como "anda" e "gira" (ações estas que fazem parte do repertório de ações básicas conhecidas pelo ator "caneta"). Esta versão continha o "palco da caneta", no qual se podia controlar o objeto "caneta" da mesma forma que a tartaruga *Logo*. Ao

interagir com o ator "caneta" o aprendiz estará exercitando todo um conjunto de noções básicas de programação de computadores, sendo que estas estão ajustadas em um ambiente de POO (Mariani, 1998).

Para trabalhar com o palco caneta, o aprendiz deve trocar o palco, selecionando o *PalcoDaCaneta* conforme indicado nas figura 2. É apresentado então, o elemento virtual caneta representado por uma seta. Para interagir com ela utiliza-se de comandos escritos na área de comandos (figura 3), ou um menu que apresenta algumas mensagens que podem ser enviadas para a caneta (figura 4).

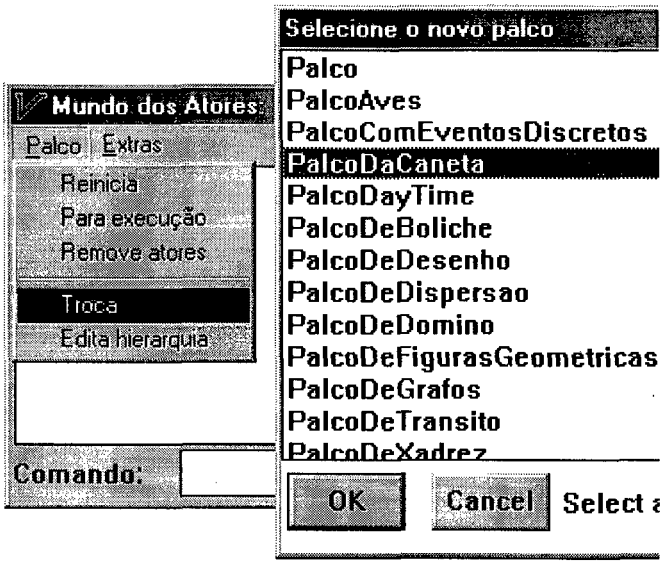

Figura 2: Seleção do PalcoDaCaneta

Figura 3: Exemplo de comando para a caneta

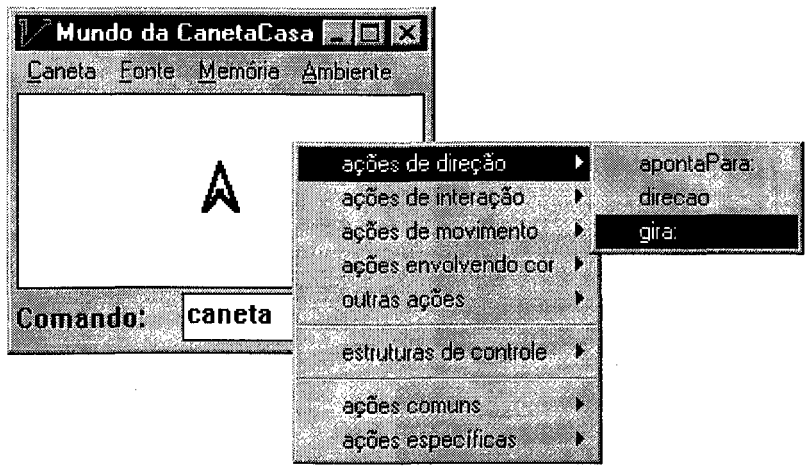

Figura 4: Ações disponíveis para interação com a caneta

Pode-se considerar que os atores se comportam como a tartaruga de *Logo,* devido a algumas características como, possuir uma representação gráfica, uma posição no palco, uma direção para movimentação, capacidade de movimento para frente e giro. Além destes, os atores possuem características que são consideradas como não existentes em *Logo,* como a capacidade de sentir se o ator está sendo tocado por outro ou mesmo se existe algum ator nas proximidades (Wazlawick et al, 2001).

O papel que cada ator desempenha é especificado por um conjunto de sentenças no método "`definePapel`". Cinco tipos básicos de sentenças são possíveis e citadas por Wazlawick et al (2001):

a) **Ação incondicional**: esta ação é definida pelo método `sempreFaça:`. Por exemplo: "`ator sempreFaça: [ator anda: 10; gira: 5]`" (o ator a cada instante anda 10 passos e gira 5 graus, descrevendo um círculo ao longo do tempo).

b) **Ação condicional**: esta ação é definida pelo método "`sempreQue:faça:`". Por exemplo: "`ator sempreQue: [ator estáSendoTocado] faça: [ator anda: 100]`" (o ator anda 100 passos para frente sempre que outro ator tocar nele).

c) **Ações de intervalo fixo**: estas ações são definidas pelo método "`cada:faça:`". Por exemplo, "`ator cada: 30 faça: [ator anda: 10]`" (a cada 30 instantes de tempo o ator anda 10 passos para frente).

d) **Ações probabilísticas**: estas ações são definidas pelo método "`comProbabilidadeDe:faça:`". Por exemplo: "`ator comProbabilidadeDe: 0.2 faça: [ator gira: 30]`" (a cada instante o ator pode girar 30 graus com probabilidade de 20%).

e) **Ações em instantes de tempo específicos**: são definidas pelo método "`em:faça:`". Por exemplo, "`ator em: 1000 faça: [ator deixaPalco]`" (após 1000 instantes contados a partir da definição deste comportamento, o ator sai do palco).

Na criação de um novo palco ou especialização de um palco já existente, os comportamentos dos objetos devem ser programados em linhas de códigos como mostra a figura 5. Pode-se utilizar das ações já programadas em cada palco, disponíveis e conhecidas pelos atores, ou incorporar novas ações facilmente, para que o palco possa ser modificado para se adequar a uma nova situação.

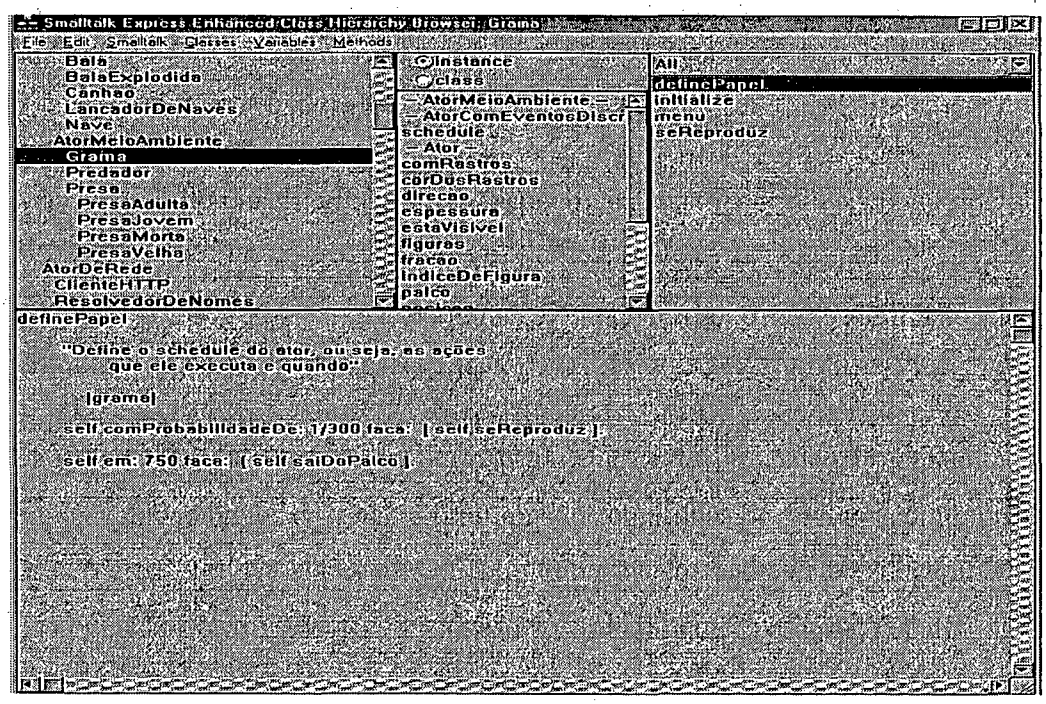

Figura 5: Interface fornecida para programação de comportamentos

3.1 Aplicações Construídas com a Ferramenta Mundo dos Atores

A ferramenta *Mundo dos Atores* é apresentada na disciplina de programação orientada a objetos, no primeiro ano do curso de Ciência da Computação da UFSC (Universidade Federal de Santa Catarina), sendo aplicada também em diversos outros cursos no Brasil. A metodologia utilizada propõe que os conceitos sejam vistos em paralelo, e não do mais básico para o mais complexo, através do ensino dirigido por problema ou estudo de caso.

Após um trabalho introdutório aos conceitos necessários para o trabalho com a ferramenta, é solicitado aos estudantes o desenvolvimento de aplicações (Wazlawick et al, 2001). São desenvolvidos trabalhos coletivos (Wazlawick & Mariani, 2000), sendo que, logo após é solicitado aos estudantes que proponham seus próprios projetos. Na avaliação dos projetos propostos são observados alguns aspectos pelo professor considerados relevantes. Dentre eles pode-se citar o tempo disponível para elaboração do projeto e conceitos de orientação a objetos a serem aplicados, satisfazendo desta forma o objetivo da disciplina.

Segundo Wazlawick et al (2001), após a experiência realizada com o Mundo dos Atores por alguns anos, envolvendo cerca de 300 estudantes, ficou evidente que cada projeto desenvolvido utilizava conceitos de orientação a objetos aprendidos de forma paralela e dirigida por projeto. Os estudantes provaram ser capazes de compreender conceitos novos rapidamente, e mesmo quando implementavam suas primeiras versões de forma incorreta ou inconveniente, foram capazes de melhorar as abordagens.

Pela ferramenta estar vinculada a uma linguagem de programação (*smaltalk*), foram observadas algumas dificuldades no uso da sintaxe da linguagem. Sendo que, este fato não invalida o uso da ferramenta visto que seu objetivo é justamente permitir a aprendizagem de uma linguagem de programação orientada a objetos.

A seção a seguir apresenta um relato desta experiência da ferramenta com alunos, através de alguns projetos elaborados pelos mesmos.

3.1.1 Projetos dos Estudantes

O objetivo da aplicação da ferramenta Mundo dos Atores é permitir que o estudante aprenda conceitos de programação orientada a objetos, através do ensino dirigido por problema ou estudo de caso. Como os próprios alunos formulam seus projetos, não há tema preestabelecido e cabe a eles a escolha do tema desde que, o tema aborde os conceitos a serem estudados na disciplina. Observa-se que, na maioria dos trabalhos escolhidos o tema envolvia jogos (Wazlawick et al, 2001). As seções subseqüentes apresentam uma amostragem destes projetos e exemplos de soluções de programação criadas pelos estudantes. Algumas destas soluções podem ser encontradas em Wazlawick et al (2001).

3.1.1.1 Projeto Invasores do Espaço

Invasores do espaço é um jogo que parte da idéia da existência de uma nave-mãe que lança naves menores. Cada nave menor lança bombas que caem em direção à parte inferior da tela, dando a idéia que estas estão caindo para o chão, onde um canhão controlado pelo usuário está localizado. O usuário é capaz de mover o canhão para a esquerda e para a direita, e disparar mísseis. Um único míssil pode destruir uma bomba ou uma nave menor, e uma quantidade grande de mísseis é necessária para destruir uma nave mãe.

As classes de atores envolvidas na solução do problema foram as seguintes: `Canhão`, `CanhãoExplodido`, `NaveMãe`, `NaveMãeExplodida`, `Míssil`, `MíssilExplodido`, `Bomba`, `BombaExplodida`, `Nave` e `NaveExplodida`. A identificação destas foi facilitada pela ocorrência destes objetos como entidades gráficas no problema.

O canhão não pode ser movido para fora do palco. Assim, quando ele está fora do palco, ele anda para trás de volta ao palco, e quando ele é tocado por uma instância de `BombaExplodida` ou `NaveExplodida`, ele se torna um `CanhãoExplodido`:

```
definePapel "na classe Canhão"
ator sempreQue: [ator foraDoPalco]
     faça: [ator gira: 180; anda: 35].
```

```
ator sempreQue: [ator tocaAtorDoTipo: BombaExplodida]
     faça: [ator tornaSe: CanhãoExplodido].
ator sempreQue: [ator tocaAtorDoTipo: NaveExplodida]
     faça: [ator tornaSe: CanhãoExplodido]
```

O comportamento de um canhão explodido deve ser a sua existência por uma breve fração de tempo após sua criação e então desaparecer:

```
definePapel "na classe CanhãoExplodido"
ator em: 10 faça: [ator saiDoPalco].
```

A nave mãe de tempos em tempos pula para uma região diferente do palco. Da mesma forma, em determinados tempos ela lança uma nave menor. Ao tocar qualquer objeto, ela diminui o valor de seus escudos. Zerando o valor dos seus escudos, ela se torna uma `NaveMãeExplodida`:

```
definePapel "na classe NaveMãe"
ator cada: 40 faça: [ator pula].
     "o método 'pula' é implementado a parte"
ator cada: 15 faça: [ator lancaNave].
     "lancaNave é definido abaixo"
ator sempreQue: [ator tocaUmAtor]
     faça: [escudos := escudos - 1].
ator sempreQue: [escudos <= 0]
     faça: [ator tornaSe: NaveMãeExplodida].
```

O método `lancaNave` é implementado da seguinte forma:

```
lancaNave "na classe "NaveMãe"
| nave |
nave := Nave nova.
nave entraNoPalco: ator palco.
     "a nave entra no mesmo palco da nave mãe"
nave pulaPara: ator posicao.
     "a nave vai para a posição da nave mãe"
```

Uma nave ainda deve mover-se pelo palco, soltar bombas, sentir se foi tocada pelo chão ou por um míssil explodido. Ela torna-se uma nave explodida, se for tocada por um míssil explodido:

```
definePapel "na classe Nave"
ator sempreFaça: [ator anda: 4].
ator comProbabilidadeDe: 0.2 faça: [ator soltaBomba].
"a implementação de solta bomba é similar à de lança nave"
ator sempreQue: [ator atingiuOChão ]
     faça: [ator tornaSe: NaveExplodida].
     "atingiuOChão é implementado como um teste de posição".
ator sempreQue: [ator tocaAtorDoTipo: MíssilExplodido]
     faça: [ator tornaSe: NaveExplodida].
```

Este projeto foi desenvolvido pelos alunos em poucas semanas (Wazlawick et al, 2001). O resultado deste jogo é apresentado na figura 6.

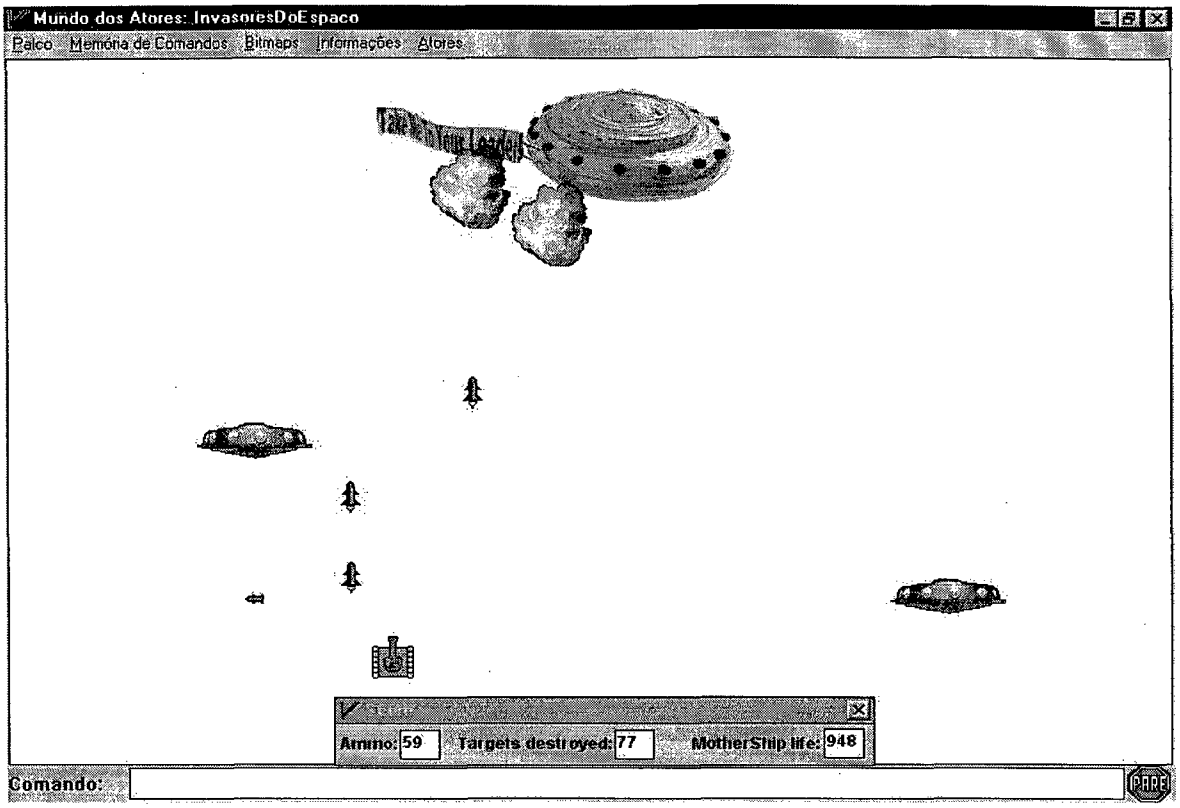

Figura 6: Interface do Jogo Invasores do Espaço

3.1.1.2 Projeto Meio Ambiente

O projeto Meio Ambiente considera um ecossistema habitado por elementos como: `Predador`, `Presa Adulta`, `Presa Jovem` e `Grama`.

O papel da `Grama` consiste em reproduzir-se a cada 300 unidades de tempo e morrer quando atingir 750 unidades de tempo de vida, como demonstrado no código abaixo:

```
definePapel "na classe Grama"
ator cada: 300 faça: [
```

```
      grama := Grama new.
      ator adiciona: grama.
      grama
            pulaPara: ator posicao;
            apontaPara: (ator aleatorio:360);
            anda: ator tamanho x.
].
ator em:750 faça: [ator destroi].
```

A PresaJovem, deve adotar uma PresaAdulta como mãe e seguí-la para onde ela for. Caso ela não tenha mãe, ela anda aleatoriamente (sem rumo) pelo ambiente. Após 1000 unidades de tempo ela passa a ser uma PresaAdulta.

```
DefinePapel
ator sempreQue: [mae isNil or[mae palco isNil]] faça: [mae: ator
            adultoMaisProximo].
ator sempreFaça:[
      mae isNil ifTrue: [
            ator gira: (ator aleatorio: 90) - 45; anda: 5.
      ] ifFalse:[
            (distancia:= ator distanciaPara: mae)> 15 ifTrue:[
                  ator apontaPara: mae; anda: (distancia - 15 min:
                      13)
                  ].
            ].
      ].
      ator em: 1000 faça: [ator tornase: PresaAdulta new].
```

O Predador inicia com um certo estoque de energia. A cada tempo discreto este nível é reduzido em uma unidade. Morre quando seu nível de energia se encontrar com valor zero e/ou após 3000 unidades de tempo. Sai a procura de caça quando seu nível de energia for menor que 700. Cada presa caçada aumenta seu nível de energia. Se estiver alimentado, se reproduz de tempos em tempos.

```
definePapel
ator sempreFaça: [nivelDeAlimentacao:= nivelDeAlimentacao - 1].
ator sempreQue: : [nivelDeAlimentacao = 0] faça: [ator destroi].
ator sempreQue: : [nivelDeAlimentacao < 700] faça: [ator caça].
ator em: 3000 faça:[ator destroi].
ator cada:1050 faça: [
      nivelDeAlimentacao > 400 ifTrue:[
      jovem:=Predador new.
      ator palco adiciona: jovem.
      Jovem pulaPara: ator posicao.
      ].
].
```

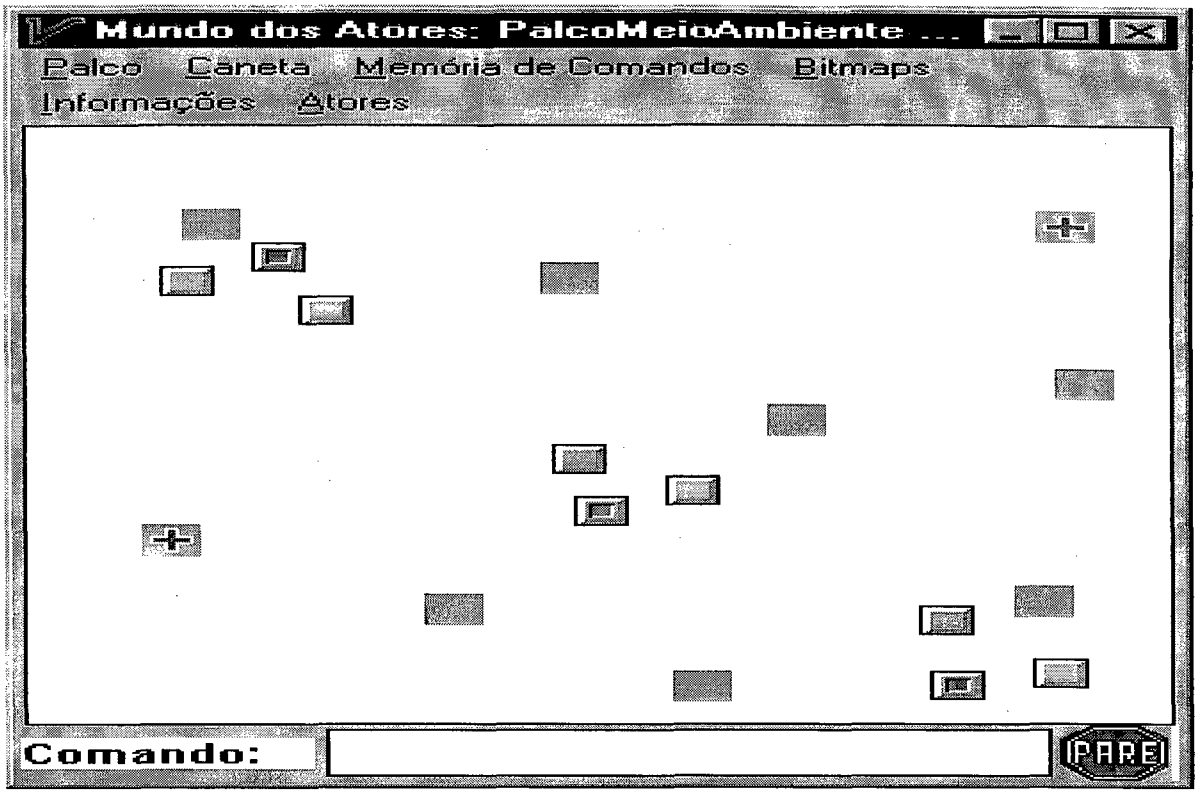

Figura 7: Palco Meio Ambiente.

3.1.1.3 Projeto Jogo da Memória

O jogo da memória pode ser jogado individualmente ou em dupla de forma alternada, ou seja, quando um jogador erra, a vez é passada para o outro. No tabuleiro são dispostos pares de figuras com a mesma imagem, inicialmente as figuras estão ocultas e distribuídas de forma aleatória. As fases apresentam conjuntos de 2, 4, 8, 16, 32, 64 e 98 pares. No tabuleiro é apresentado um placar com a pontuação de cada jogador. A jogada é realizada virando-se duas peças do tabuleiro. Se elas forem iguais o jogador ganha 1 ponto e elas permanecem nesse estado até o final da fase e o jogador continua virando. Caso as figuras viradas sejam diferentes, o jogador passa a vez e as figuras voltam a situação inicial. A fase termina quando todos os pares estiverem revelados. Entre os pares de figuras estão cinco pares de bônus. Os bônus são pares que

acrescentam pontos ao jogador que acertar esse par. Os valores são de 10, 20, 30, 40 e 50, sendo que os pontos serão computados multiplicando esse valor por 10.

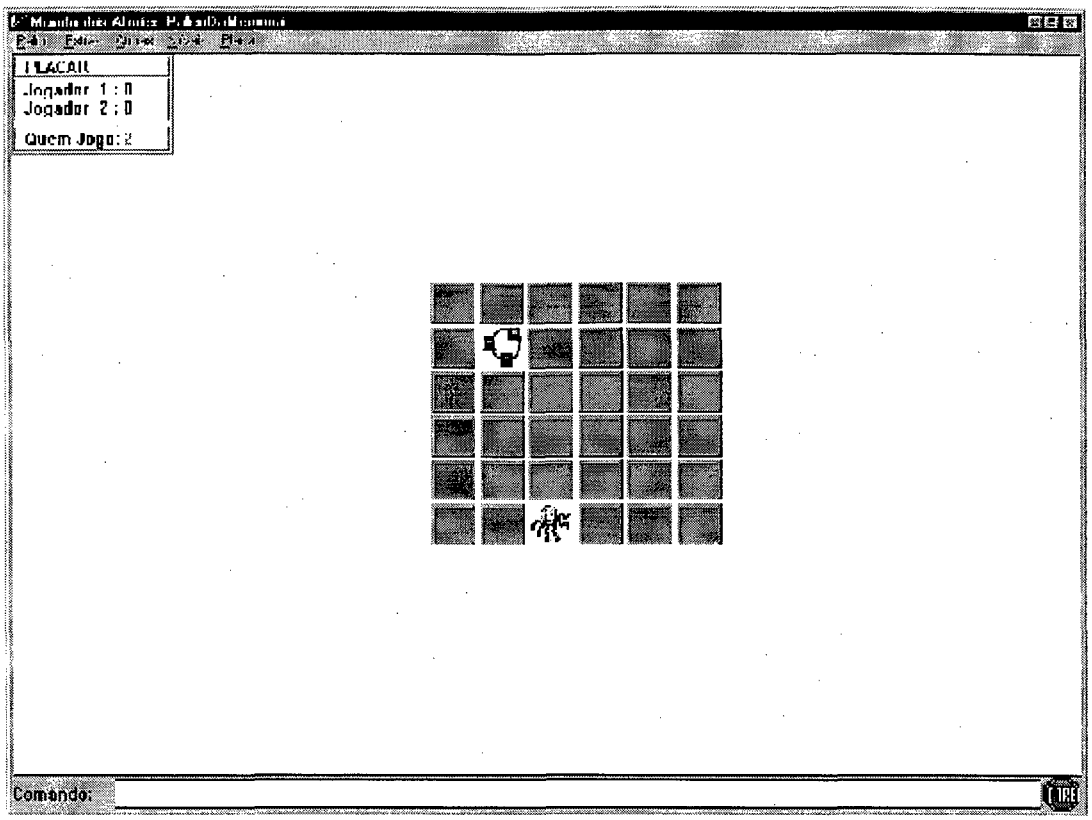

Figura 8: Interface do Jogo da Memória

3.1.1.4 Projeto Bricks

O jogo Bricks possui uma palheta horizontal na parte inferior do palco que se desloca da esquerda para a direita, sendo controlada pelo usuário usando as setas de navegação, com o objetivo de não deixar a bolinha tocar a parte inferior da tela que representa o chão. A palheta diminui de tamanho e a velocidade da bolinha aumenta a partir do momento que o jogador passar para uma nova fase.

Na parte superior do palco existe um conjunto de blocos que, ao serem tocados pela bolinha desaparecerem, marcando assim pontos para o jogador. Quando não

existirem mais blocos na parte superior do palco o jogador passa para a próxima fase do jogo. A bolinha fica navegando pelo palco, e quando tocar na parte superior e atingir um bloco, este muda a sua imagem e se movimenta de acordo com o ângulo de rebatida em direção ao chão e desaparece do palco, sendo contado ponto para o jogador. Caso ela toque em um dos lados do cenário, a mesma é rebatida, exceto quando ela bater na parte inferior do cenário (chão), o que faz o jogador perder uma das vidas que possui. O jogo é finalizado quando o jogador não possui mais vidas. A interface apresentada ao usuário pode ser vista na figura 9.

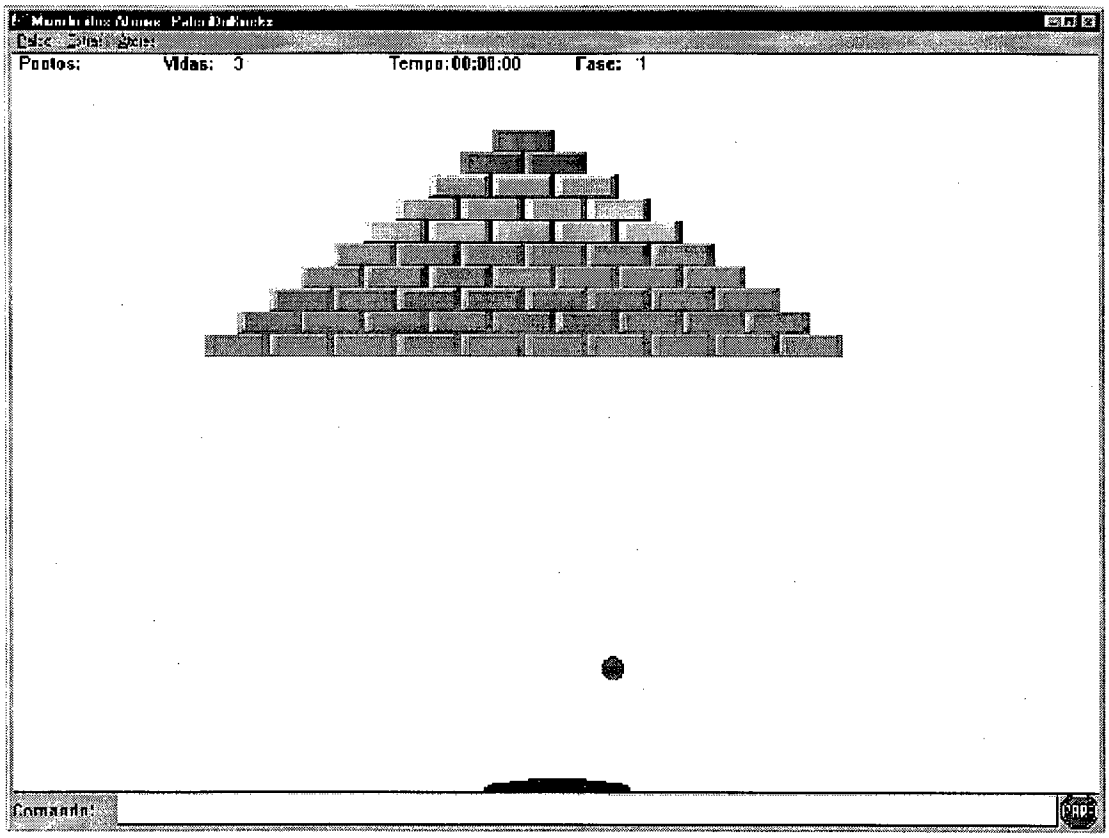

Figura 9: Interface do Projeto Bricks

3.1.1.5 Projeto Sinuca

No jogo Sinuca, as bolas são os principais atores. A bola branca além de todas as características das bolas comuns é capaz de ser movimentada por um taco. O taco é

representado por um ponto preto, e pode ser movido com o teclado através de ajuste grosso e fino. A força do taco também pode ser definida pelo teclado antes de dar a tacada. Uma vez iniciado o movimento das bolas, a inércia (quantidade de movimento) é repassada entre as bolas que se tocam de acordo com o ângulo do choque. A interface do jogo pode ser visualizada na figura 10.

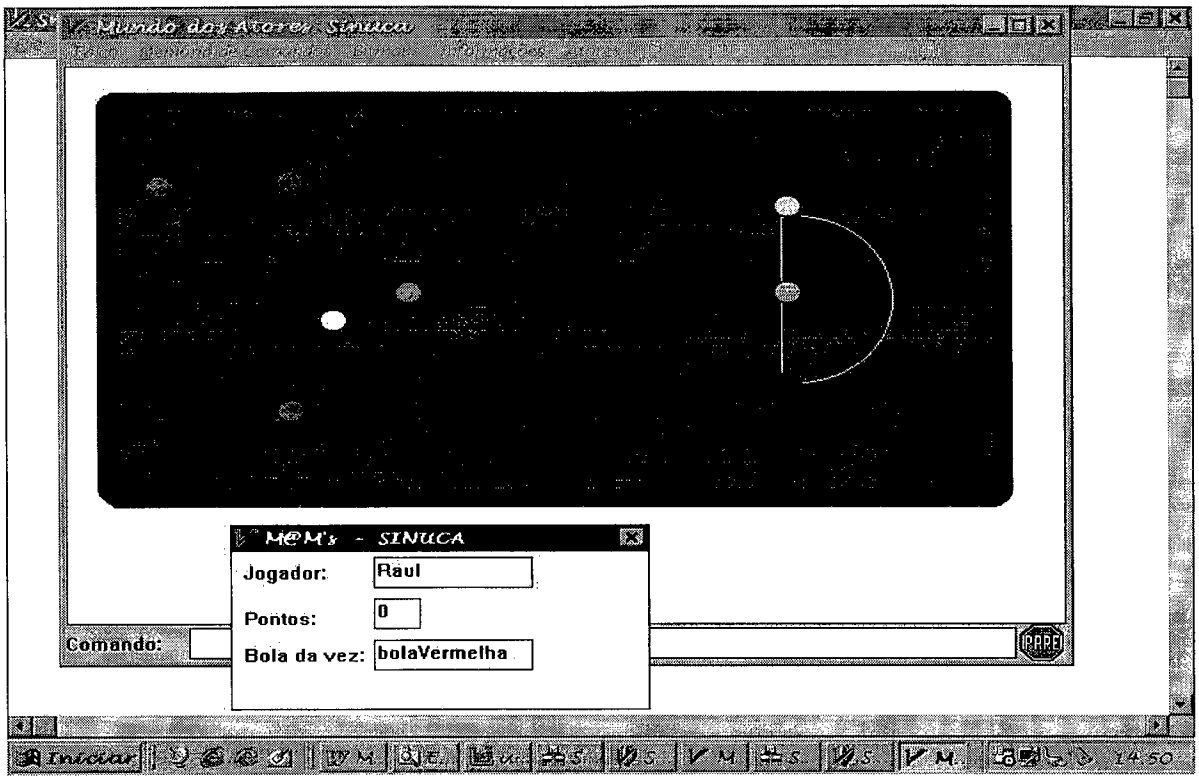

Figura 10: Interface do Jogo de Sinuca

3.1.1.6 Projeto Minhoca

Trata-se de um jogo onde uma minhoca busca alcançar seu destino (a superfície), movimentando-se com objetivo de desviar de obstáculos aleatórios e fugir de formigas. Para se proteger do ataque das formigas, a minhoca possui 04 (quatro) poções de veneno, que representam suas vidas. Ela morrerá se for atacada pelas formigas e não possuir mais veneno. São criados túneis pela minhoca quando do seu

deslocamento (figura 11), e estes serão os mesmos percorridos pelas formigas na perseguição da minhoca. As formigas são libertadas de alguns obstáculos existentes no palco e seu comportamento é apenas de perseguir os caminhos criados pela minhoca, não criando então novos caminhos.

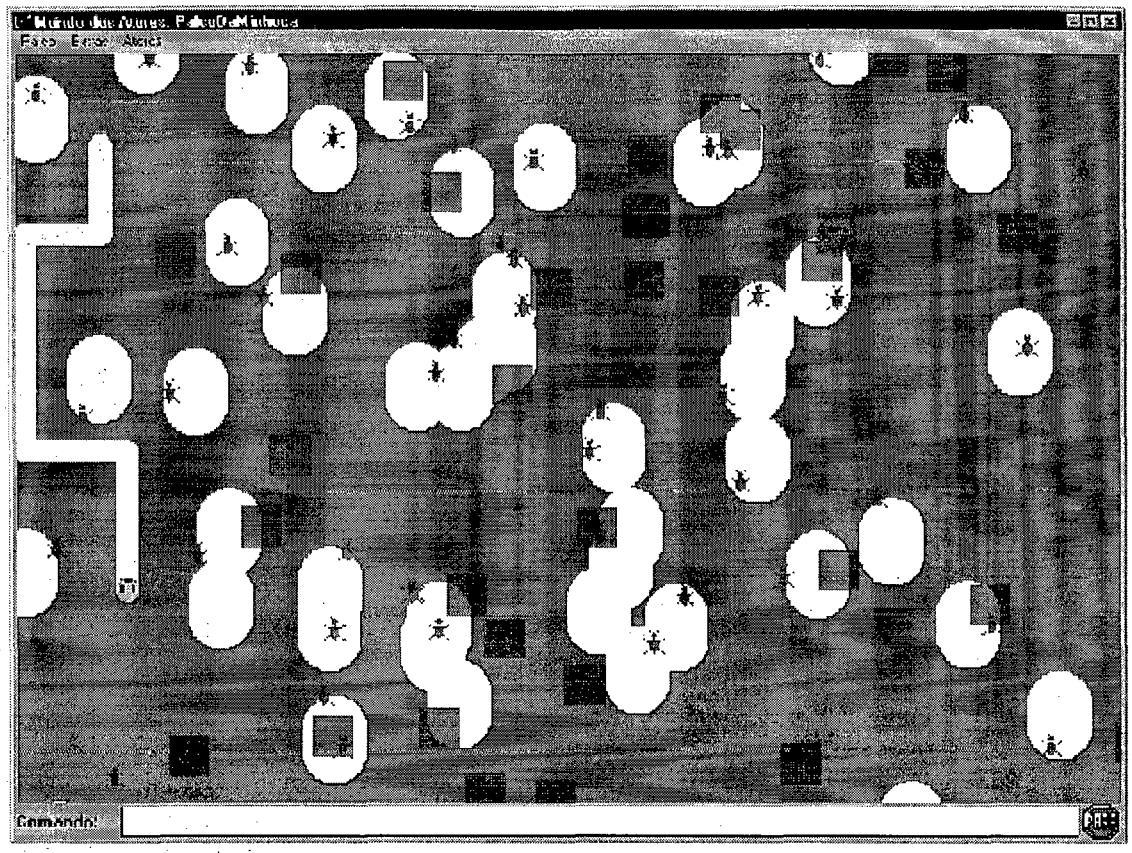

Figura 11: Interface do Projeto Minhoca

3.1.1.7 Projeto Batalha Medieval

Batalha medieval consiste em um jogo em que cada vez que a pedra vinda da catapulta atinge o castelo parte dele se desmancha. Cada tijolo do castelo é um ator. O comportamento do ator 'tijolo" consiste em verificar a cada instante se foi atingido de cheio ou de raspão por uma pedra. Se foi atingido de cheio, o tijolo se desintegra, desaparecendo do palco. Se foi atingido de raspão, ele irá se deslocar para uma posição

que dependerá do ângulo e distância do impacto. Sempre que um tijolo estiver em pleno ar, ele iniciará um processo de queda livre. Isso significa que se o castelo for atingido na parte mais baixa, os tijolos que estiverem acima do vão livre vão desmoronar de uma forma muito realista. A interface final do jogo é apresentada na figura 12.

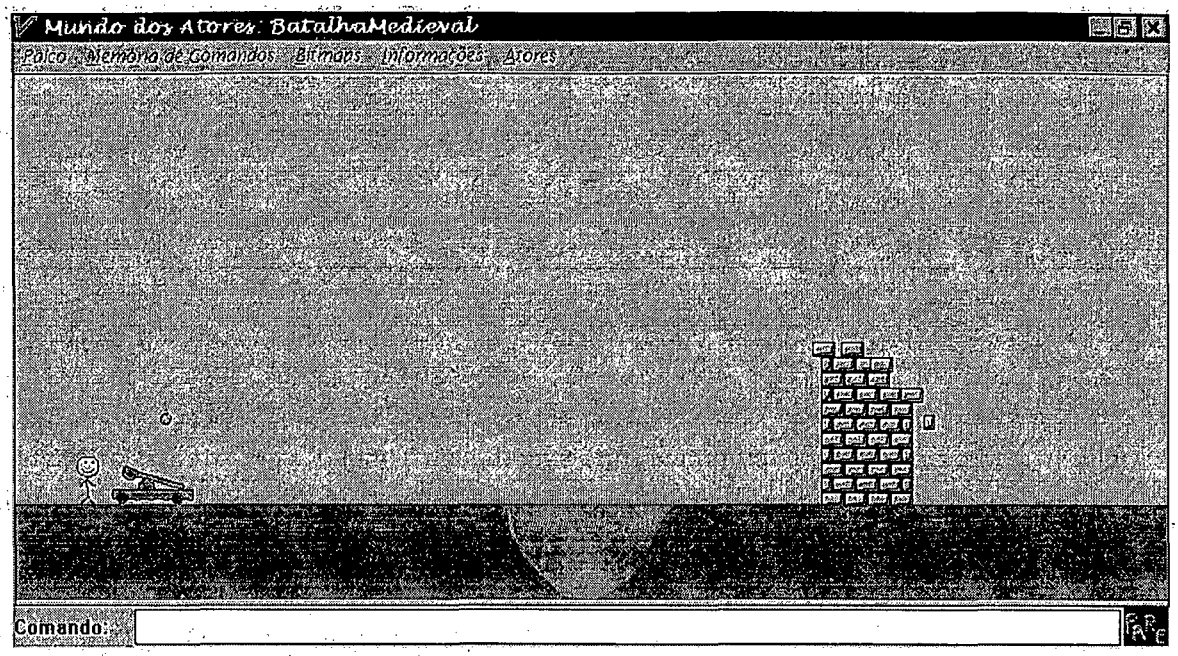

Figura 12: Interface da Batalha Medieval.

3.1.1.8 Projeto FreeWay

O objetivo deste jogo é fazer com que um Sapo atravesse uma auto-estrada com 8 pistas, quantas vezes puder dentro de um tempo estabelecido. Ganha o jogo quem fizer o Sapo ultrapassar o maior número de vezes a auto-estrada. A cada quatro pistas ultrapassadas, o sentido dos carros muda. Em cada pista os carros possuem velocidades diferentes. Em uma das pistas de cada sentido, há caminhões trafegando, com o dobro do tamanho dos carros. O Sapo pode se mover em todas as direções, inclusive voltar a uma pista que ele já ultrapassou, na tentativa de desviar dos carros e caminhões. Quando o Sapo conseguir ultrapassar a auto-estrada e chegar na calçada, somará 1 (um) ponto e

voltará à posição inicial, mas na mesma posição vertical em que estava quando alcançou a calçada. Cada vez que o Sapo for atropelado por um carro ou caminhão, ele voltará automaticamente para a pista anterior. O sapo ainda possui um tempo de vida estabelecido, esgotado este tempo o jogo termina.

3.1.1.9 Projeto River Ride

O jogo possui um avião que anda em direção ao topo da tela, o controle para movimentação é feito através das setas de navegação, deslocando-se lateralmente para a esquerda e direita em um ângulo de 90 graus. O avião atira a munição, através da barra de espaço, sempre na mesma direção do topo. Quando a munição atinge um objeto do tipo helicóptero, navio ou submarino, este desaparece e o avião recebe pontos.

O avião, quando se sobrepõem ao posto, é abastecido de combustível. Na falta de combustível o avião cai, e desaparece. À medida que o avião anda e atira, ele perde combustível.

O cenário é composto por uma figura que forma o fundo do palco. Vários elementos, entre eles helicópteros, navio, submarino se movimentam no palco, em ângulos diferenciados.

3.1.1.10 Projeto Gato & Rato

O projeto Gato & Rato é um jogo de perseguição onde, são criados inicialmente um rato e um gato. Os comportamentos destes atores são respectivamente de fuga e perseguição, sendo que o rato é controlado pelo usuário e pode mover-se em nas quatro direções no palco e, o gato move-se unicamente em movimento de perseguição ao rato

com velocidade constante. Assim que o tempo do jogo avança, são gerados mais gatos. O jogo é finalizado quando um gato consegue alcançar o rato.

4 Análise dos Projetos dos Alunos

A análise dos projetos dos alunos baseia-se na investigação de comportamentos de objetos dinâmicos existentes nas aplicações implementadas com a ferramenta Mundo dos Atores. Um dos critérios adotados para esta investigação é a análise das linhas de código correspondentes aos comportamentos dos objetos, relacionando-as com os comportamentos observados quando a aplicação está em execução.

Outro critério de investigação é a observação dos comportamentos dos objetos, quando os mesmos estão sendo executados, não considerando o código que determina suas ações. Isto possibilita a identificação e registro de comportamentos observados no decorrer do tempo de execução da aplicação.

Independente do critério de investigação a ser utilizado na aquisição de padrões de comportamentos deve-se inicialmente identificar um padrão dentre os comportamentos existente nas aplicações. No caso específico deste trabalho, não havia conhecimento prévio de possíveis padrões ou comportamentos implementados nas aplicações. Por ausência de uma metodologia de identificação de padrões de comportamentos foi criado um esquema ilustrado na figura 13 para ser usado como guia na investigação de padrões de comportamentos existentes no conjunto de aplicações disponíveis para análise.

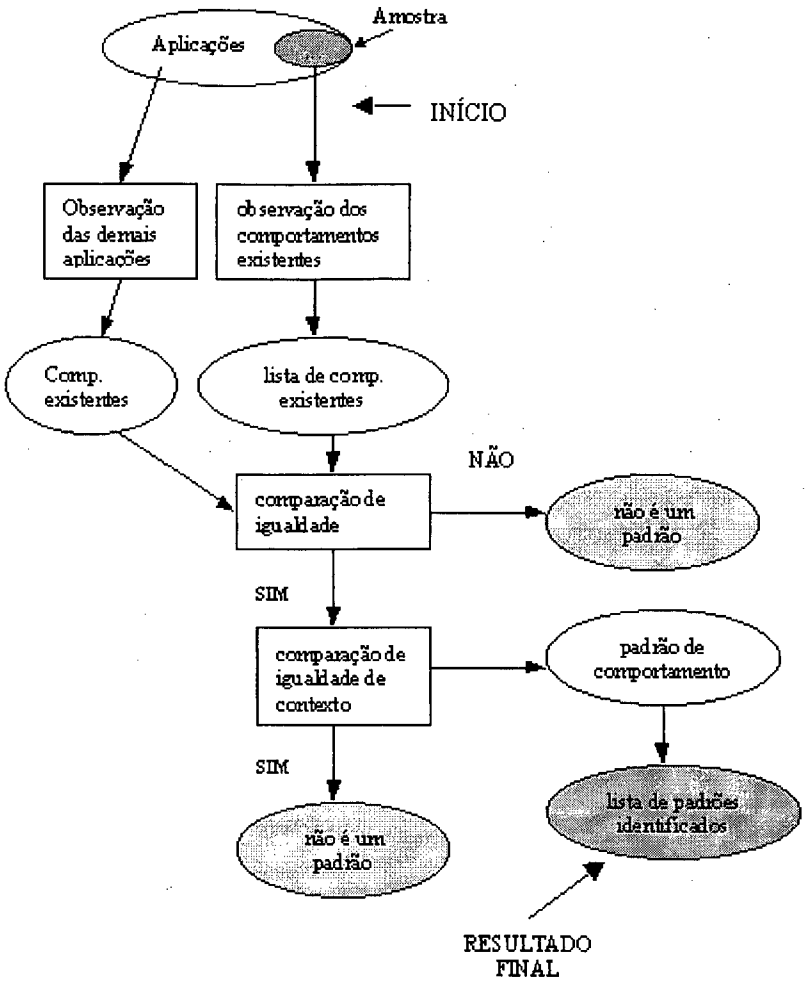

Figura 13: Esquema para reconhecimento de padrões de comportamentos

Para a identificação de um padrão dentre os comportamentos implementados partiu-se para a observação dos comportamentos existentes em uma amostra do conjunto total de aplicações. Com isto, obteve-se a identificação dos comportamentos existentes nestas aplicações. No passo seguinte foi dada continuidade na observação dos comportamentos existentes nas demais aplicações e, a medida que, os comportamentos observados coincidiam com os comportamentos já adquiridos, e estes estando presentes em um contexto diferenciado do anterior era adotada a ocorrência de um padrão de comportamento. As ocorrências de padrões de comportamentos constituíam a lista de padrões de comportamentos de objetos dinâmicos implementados nas aplicações analisadas.

Esta análise envolveu cerca de 50 projetos. A grande maioria dos quais realizados por alunos de mestrado em ciência da computação da disciplina *Sistemas Orientados a Objetos*.

4.1 Padrões de Comportamentos Identificados

Numa análise inicial dos projetos, pôde-se identificar várias situações que se repetem mesmo quando da mudança de contexto e projeto. Esta repetição, como já descrita por Alexander (1993) pode ser considerada como padrão de projeto. Neste texto, adotar-se-á a denominação de *padrão de comportamento*, pois a pesquisa trata de comportamentos de objetos dinâmicos.

O comportamento de objetos dinâmicos aqui é definido como uma série de ações simples condicionadas que devem ser executadas em cada instante de tempo. Sugere-se neste trabalho que estes comportamentos sejam descritos por meio de uma tabela condição/ação. Nesta tabela determina-se o conjunto de ações e condições básicas do comportamento, o que possibilitaria observar se alguns comportamentos de objetos possuem as mesmas condições/ações, ou variações destas.

4.1.1 **Critério de Investigação I:** *Análise do Código Fonte dos Atores e Ações Executadas no Palco*

A investigação dos comportamentos de objetos dinâmicos existentes nos projetos obedecendo este critério, possibilita encontrar comportamentos que não são percebidos quando se observa somente os atores em ação no palco (aplicação em execução). Além disto, pode-se constatar que, muitos comportamentos descritos no código fonte dos projetos não eram executados. Observou-se que este fato ocorre por estes comportamentos estarem condicionados a situações com pequena probabilidade de ocorrência, e em outros casos, situações impossíveis de ocorrer.

A aplicação deste critério de análise permite elaborar uma lista inicial de comportamentos. Depreende-se desta lista a existência de comportamentos semelhantes. Esta constatação implica num tratamento desta lista com a finalidade de identificar e destacar os comportamentos semelhantes.

A lista de padrões de comportamentos, também permite constatar que um padrão de comportamento possui alguns elementos, e que este pode ser dividido em um *ativamento* e uma *ação*. Onde o *ativamento* seria uma condição ou regra preestabelecida que o ator deve cumprir para acionar o comportamento. Já a *ação* seria o comportamento desempenhado pelo ator no palco. Os conceitos de *ativamento* e *ação* possibilitam dividir a lista de padrões de comportamento em duas novas listas, uma de *ativamentos* e outra de *ações*, apresentadas nas tabelas 1 e 2 respectivamente.

Tabela 1. – Ativamentos dos comportamentos identificados.

Cada < `número inteiro` > instantes de tempo o ator < *ação* >
Com uma probabilidade de < `número real` > o ator < *ação* >
Em < `número inteiro` > instantes após ser criado o ator < *ação* >
Sempre que o ator estiver com a imagem < `figura` > o ator < *ação* >
Sempre que o ator se encontrar fora do palco o ator < *ação* >
Sempre que o ator se encontrar a uma distância menor que < `número inteiro` > do < `ator` > o ator < *ação* >
Sempre que o ator colidir com < `ator` > o ator < *ação* >

Tabela 2. – Ações dos comportamentos identificados.

< *ativamento* > **o ator anda** < `número real` >.
< *ativamento* > **o ator muda sua direção aleatoriamente.**
< *ativamento* > **o ator muda sua imagem.**
< *ativamento* > **o ator cria um outro exemplar da classe** < `ator` >.
< *ativamento* > **o ator se destrói.**
< *ativamento* > **o ator encerra a execução do palco.**
< *ativamento* > **o ator muda sua imagem para** < `imagem` >.
< *ativamento* > **o ator aponta para a direção do** < `ator` >.
< *ativamento* > **o ator aponta para a direção oposta ao** < `ator` >.

Onde

< *ação* > : Uma *ação* da lista de *ações* dos comportamentos encontrados. Porém, nem toda *ação* se encaixa em todo *ativamento*, um exemplo disto é mostrado nas seguintes sentenças:

Ativamento:

`Sempre que` **o ator se encontrar fora do palco o ator** < *ação* >

Ação:

< *ativamento* > **o ator muda sua imagem para** < `imagem` >

ou

< *ativamento* > **o ator aponta para a direção oposta ao** < `ator` >

Neste exemplo não faz sentido o ator mudar sua imagem ou apontar para a direção oposta de outro ator se estiver fora do placo. Se o ator estiver fora do palco não estará mais visível e então, não poderá ser visualizada a nova imagem atribuída a ele, e nem a direção para onde está apontando.

< `número inteiro` > : Um número inteiro natural.
< `número real` > : Um número real.
< `figura` > : Uma das figuras do ator em questão.
< `ator` > : Um ator que se encaixe ao *ativamento*.
< *ativamento* > : Um *ativamento* da lista de *ativamentos* dos comportamentos encontrados.

Os comportamentos identificados nesta análise podem ser considerados comportamentos muito simples, e em alguns casos encontrados apenas na investigação do código fonte dos comportamentos dos atores. Por estes fatos, o Critério de Investigação I sozinho é inadequado para encontrar comportamentos genéricos que podem ser observados em diferentes projetos, e aplicados em diferentes contextos independentemente de valores de parâmetros.

A forma de catalogação obtida nesta investigação assemelha-se com a maneira que os comportamentos são atualmente implementados. Desta forma, o desenvolvedor continua sujeito às linhas de códigos na implementação dos comportamentos, e dependente de sentenças como: `cada, em, sempre que, com probabilidade de,` entre outras, que são usadas na ferramenta Mundo dos Atores para a construção da lógica de funcionamento dos comportamentos.

Mesmo assim, a combinação destes comportamentos simples identificados neste critério de investigação tomando como base o *ativamento* e a *ação*, pode resultar na criação de comportamentos complexos com possibilidade de inclusão a atores em tempo de execução. Por exemplo, o comportamento de um objeto, que se movimenta no palco e a cada 50 instantes de tempo muda sua direção aleatoriamente, teria as seguintes sentenças:

Ativamento: `cada <número inteiro>` **o ator** `<ação>`

Ação: `<ativamento>` **o ator muda sua direção aleatoriamente**

Substituindo as sentenças pelos valores conhecidos tem-se:

`Cada <50>` **instantes de tempo o ator** `<muda sua direção aleatoriamente >`

Uma forma de facilitar a atribuição dos comportamentos adquiridos nesta análise a objetos dinâmicos sem a necessidade de programar linhas de código, é a disponibilização destes numa biblioteca com uma interface gráfica. Nesta interface o desenvolver deve escolher os *ativamentos* e as *ações* predeterminados, além de preencher os valores dos parâmetros exigidos para a ocorrência do comportamento como mostrado na figura 14 (Silva, 2001). Nesta abordagem, novos *ativamentos* e *ações* podem ser acrescentados à biblioteca, possibilitando a modelagem de comportamentos que não foram modelados ou previstos.

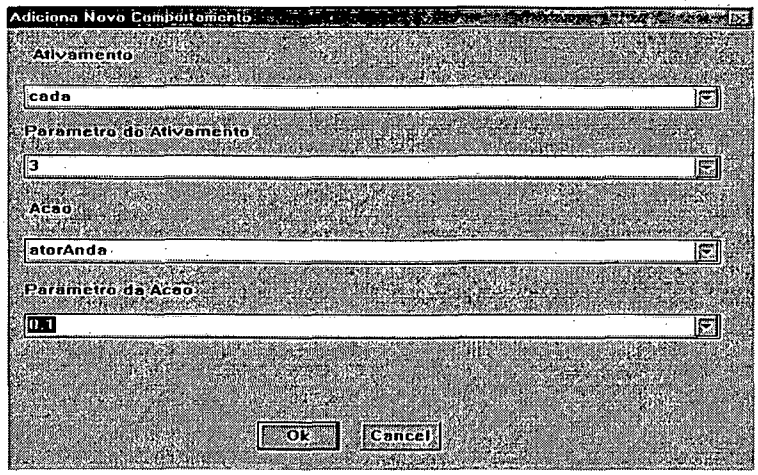

Figura 14: Interface disponível para programação de comportamentos

Com as listas de *ativamentos* e *ações* pode-se criar diferentes comportamentos simples e com a composição destes, pode-se criar comportamentos complexos como *perseguir*, *fugir*, entre outros para o mesmo objeto. Desta forma, assume-se que os objetos podem possuir uma coleção própria de comportamentos e novos podem ser inseridos em tempo de execução. Para edição da coleção de comportamentos do objeto, propõe-se uma interface ilustrada na figura 15 (Silva, 2001). Com esta interface pode-se modificar a coleção de comportamentos de determinado objeto em tempo de execução, e ajustar estes quando necessário.

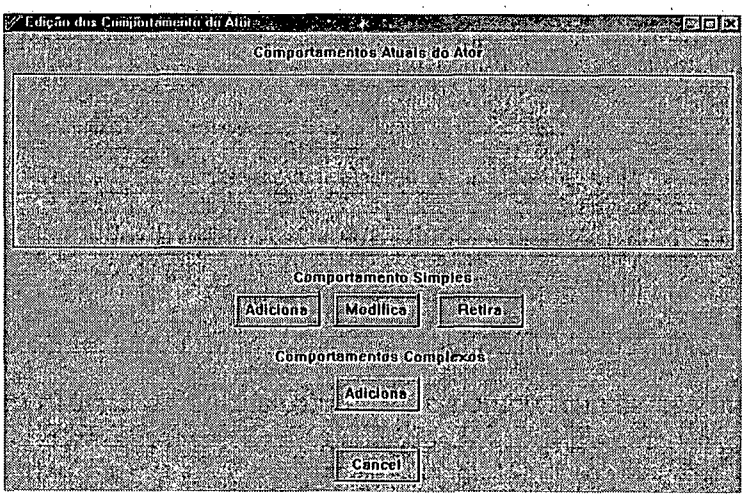

Figura 15: Interface proposta para a edição de comportamentos

Deste primeiro critério de investigação, conclui-se que o *ativamento* e a *ação* podem fazer parte da implementação de comportamentos de mais alto nível. Estes comportamentos, por sua vez, são observados na execução dos atores no palco. Outra conclusão é que os comportamentos complexos podem ser gerados através de combinação de *ativamentos* e *ações*.

Destaca-se, também, o fato que a proposta de atribuição de comportamentos através desta biblioteca não isenta o usuário de elaborar a lógica de funcionamento do comportamento. Esta abordagem não atende a proposta inicial que é indicar apenas um comportamento de alto nível para o objeto e valores de parâmetros necessários para sua ocorrência, sem se preocupar com sua implementação. Por exemplo: `perseguir(rato, 5)`, onde rato é o objeto ou classe a ser perseguida e 5 o valor para velocidade de deslocamento.

Esta carência pode ser suprida pelo próximo critério de investigação, pois se entende que não se atendo ao código do comportamento, pode-se ter uma percepção mais clara dos comportamentos de alto nível. Este critério de investigação é apresentado na próxima seção.

4.1.2 **Critério de Investigação II:** *Análise Apenas das Ações Executadas no Palco*

Neste critério, são considerados apenas os comportamentos ou ações existentes no momento da execução da aplicação. Além da descrição textual do comportamento, apresentada no ANEXO 1, determina-se parâmetros considerados essenciais e parâmetros considerados opcionais para sua ocorrência. Por outro lado, não há preocupação como tais comportamentos são ou poderiam ser implementados. Nesta investigação, obtêm-se comportamentos de mais alto nível, podendo considerá-los como encapsulamento de comportamentos encontrados na primeira investigação.

Um exemplo de situação encontrada em alguns projetos é um objeto que gera outros objetos que são impelidos numa direção aleatória. Este "gerador" se movimenta da parte inferior central da tela até o meio desta, gerando um número de "filhos" que são impelidos numa direção aleatória. Após a geração dos "filhos" ele se movimenta

numa direção até encontrar o final do Palco. Ao encontrar o final do palco ele reinicia a seqüência de ações descrita anteriormente numa direção qualquer. Os "filhos" gerados movimentam-se na mesma direção até o limite do palco e desaparecem.

Neste exemplo, pode-se identificar a existência de alguns comportamentos genéricos e os respectivos parâmetros necessários para a sua ocorrência. A descrição dos comportamentos adota a seguinte simbologia:

NomedoComportamento(`NomedoParamêtro:TipodoParâmetro`)

Abaixo são apresentados alguns dos comportamento identificados no exemplo citado nesta fase da análise:

a) **PosiçãoDireção** (`posiçãoInicial:Ponto, direção:Ângulo`) - objeto que inicialmente localiza-se em uma determinada posição definida por `posiçãoInicial`, apontando para a direção indicada por `direção` ;

b) **Gerador** (`filho:Classe,freqüência:Natural, exemplares:Número`) - consiste em gerar `Número` exemplares do objeto da classe `filho` a cada `freqüência` instantes de tempo. Os filhos são gerados na posição e direção do gerador;

c) **GeradorDireçãoAleatória** (`filho:Classe, freqüência:Natural, exemplares:Número`) - consiste em gerar `Número` exemplares do objeto da classe `filho` a cada `freqüência` instantes de tempo. Os filhos são gerados na posição do gerador numa direção aleatória;

d) **BateEVolta** (`velocidade:Número, ânguloDeRebatida:Ângulo`) - o objeto se movimenta no palco com uma velocidade indicada por `Número` e, sempre que o objeto bater em um dos limites do palco, este rebate de acordo com o `ânguloDeRebatida`;

e) **AndaEMorre** (`velocidade:Número`) - um objeto que movimenta-se sempre na mesma direção com `velocidade`, e quando chega no final do palco desaparece.

A tabela 3 apresenta a proposta de implementação dos comportamentos do "gerador" e dos "filhos" utilizando estes comportamentos.

Tabela 3. – Exemplo de implementação utilizando comportamentos identificados

Objeto	Código

"Gerador"	`DireçãoPosição(0@0, 90)`
	`GeradorDireçãoAleatória(Filho,10,1)`
	`BateEVolta(2,180)`

"Filho"	`AndaEMorre(2)`

A descrição de alguns dos padrões de comportamentos encontrados, juntamente com os parâmetros exigidos para a ocorrência dos mesmos estão na lista no ANEXO 2. Muitos dos padrões identificados neste critério de investigação assemelham-se entre si, podendo ser considerados especializações ou generalizações de outros.

Para organizá-los na forma de uma rede de comportamentos, faz-se necessária a construção do modelo conceitual de tais comportamentos. Isto significa decomposição do domínio do problema identificando os conceitos, os atributos e as associações no domínio. Para isso, pode-se utilizar notações na forma de diagramas de estrutura estática, disponibilizados em UML (*Unified Modeling Language*), (Erickson & Penker, 1998). Estas questões são abordadas no próximo capítulo.

5 Construção de um Modelo Conceitual dos Padrões de Comportamentos Identificados

A categorização de conceitos de comportamentos é necessária para a hierarquização destes, com o propósito de disponibilizar um conjunto de comportamentos *base*. Estas categorias devem ser trabalhadas no sentido de identificar agregações, generalizações e especializações dos comportamentos. Assim, torna-se possível reconhecer as semelhanças entre comportamentos, definir relacionamentos entre supertipos (conceito geral) e subtipos (conceitos especializados), e representá-los na forma de diagramas UML.

Neste capítulo, apresenta-se a construção do modelo conceitual dos padrões de comportamentos, baseando-se na categorização dos comportamentos. Para tanto, os itens a seguir tratam de conceitos relacionados à construção da biblioteca de padrões de comportamentos, além disto abordam-se ainda questões envolvidas na aplicação destes comportamentos.

5.1 Comportamentos *Base*

Comportamentos *base* são definidos como comportamentos simples, localizados em uma camada superior aos comandos da linguagem, que não necessitam de outros comportamentos para completar sua ação. Estes são aplicados a uma situação genérica e servem como ponto de partida para a geração de outros comportamentos mais complexos, com finalidade específica.

Analisando-se a lista de comportamentos identificados nos projetos com o objetivo de encontrar os comportamentos base, ver ANEXO 1, depreende-se a existência de comportamentos com características semelhantes. Esta constatação implica num tratamento desta lista com a finalidade de identificar e destacar as semelhanças, organizando-a na forma de comportamentos genéricos.

Após identificar os aspectos semelhantes e características repetitivas existentes no conjunto de comportamentos, pôde-se criar um modelo conceitual inicial. Neste modelo, foram identificadas generalizações e especializações de categoria de comportamentos, possibilitando a construção de um diagrama de estrutura estática. A cada análise deste, tem-se um novo diagrama, resultando na identificação dos comportamentos (representados por classes) constituintes da figura 16, como sendo os comportamentos *base* da biblioteca proposta.

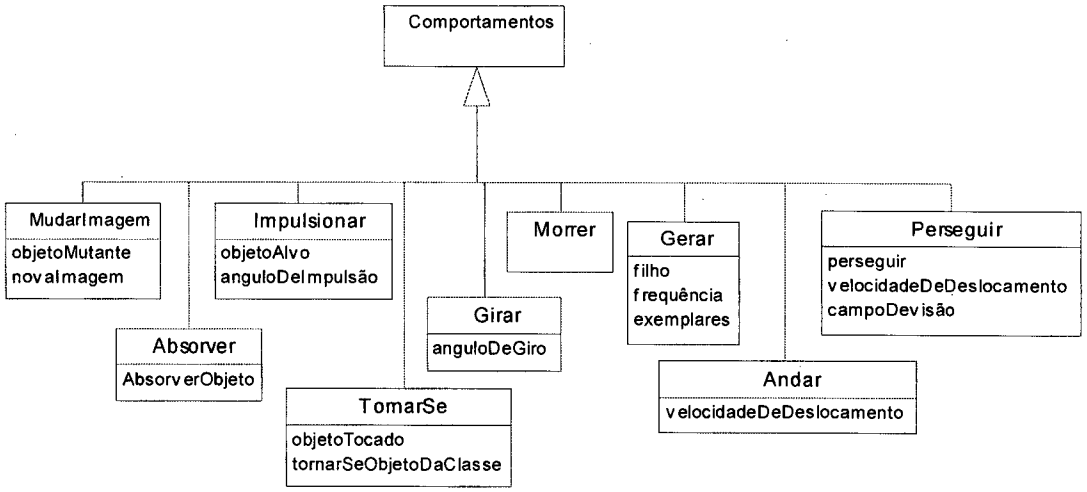

Figura 16: Diagrama de estrutura estática (classes) dos comportamentos base

Onde:

Absorver(`absorveObjeto:Classe`) – o objeto ao tocar qualquer outro objeto indicado pela `Classe`, absorve-o sendo que o absorvido desaparece.

Andar(`velocidadeDeDeslocamento:Número1`) – o objeto se movimenta com velocidade indicada por `Número1`, sempre na mesma direção até que encontre os limites do palco.

Gerar(`filho:Classe, frequência:Número2, exemplares:Número1`) – consiste em gerar `Número1` exemplares do objeto `Classe` a cada `Número1` instantes de tempo. Os exemplares são gerados na posição e direção do gerador.

Girar(`anguloDeGiro:Ângulo`) – o objeto muda sua direção obedecendo `Ângulo`.

Impulsionar(`impulsionarObjeto:Classe`) – ao tocar um objeto `Classe`, este é empurrado em direção contrária.

Morrer - o objeto desaparece.

MudarImagem(`objetoMutante:Classe, novaImagem:bitmap`) – o objeto `Classe`, troca a sua imagem atual para `bitmap`.

Perseguir(`perseguir:Classe, velocidadeDeDeslocamento:Número1, campoDevisão:Número2`) – consiste em perseguir o objeto `Classe` que está a uma distância igual ou menor que `Número2` com velocidade `Número1`.

TornarSe(`objetoTocado:Classe, tornarSeObjetoDaClasse:Classe1`) – ao tocar objeto `Classe`, este transforma-se em um `Classe1`, assumindo as características desta nova classe.

Em alguns dos comportamentos definidos como base não se consegue definir claramente parâmetros genéricos para eles (exemplo comportamento `Morrer`), ou ainda uma situação genérica para sua ocorrência (exemplo comportamento `Girar` e `MudarImagem`). Estes comportamentos no entanto fazem sentido e são essenciais na combinação com outros para a produção de comportamentos diferentes.

A combinação dos comportamentos *base* favorece a criação de outros comportamentos. Estes, por sua vez, podem ser utilizados na modelagem dos

comportamentos encontrados na fase de análise dos projetos. O item a seguir, aborda possíveis formas de criação de novos comportamentos com os já existentes.

5.2 Abordagens para a Criação de Comportamentos Complexos

Os comportamentos que contém outros comportamentos na sua formação são chamados *comportamentos complexos*, incorrendo em uma hierarquia de composição. Estes podem ser gerados utilizando-se os comportamentos *base*. Em determinadas aplicações, um comportamento independentemente de ser *complexo* ou *base* pode participar na composição de outros comportamentos.

As abordagens para a criação de comportamentos a partir dos já existentes são relatadas nos itens a seguir. Dentre estas pode-se citar a herança, a composição e a especialização de parâmetros (Larman, 2000).

5.2.1 Herança

A abordagem de herança permite que um novo comportamento seja criado a partir de outro comportamento já existente (reutilização). O comportamento *filho* herda as características do comportamento *pai*, sendo que novas características podem ser adicionadas ao novo comportamento, e este pode redefinir suas ações desempenhando seu papel de forma diferente do original (Pressman, 1995).

A herança pode ocorrer de forma *simples* (ou *hierárquica*), onde o comportamento herda as propriedades de sua superclasse em uma linha hierárquica (Ricarte, 2001). As propriedades definidas em nível mais baixo se sobrepõem ou refinam as mesmas propriedades herdadas da superclasse.

Outra forma de herança é a chamada *múltipla* (ou *reticulada*), na qual o comportamento pode herdar as propriedades de várias superclasses não relacionadas hierarquicamente. Pode-se ainda selecionar características de uma ou mais superclasses para compor o novo comportamento. Este tipo de herança é chamada de *seletiva*

(Rumbaugh, 1991). A seguir apresenta-se exemplos de alguns comportamentos criados a partir desta abordagem.

Exemplo 1: Criação do comportamento Ventilar

Ventilar (`objetoAlvo:Classe,`
`anguloDeImpulsão:Ângulo,`
`campoDeVisão: Número1`)

O comportamento `Ventilar`, consiste em deslocar objetos pertencentes à `Classe`, estando estes a uma distância menor ou igual a `Número1` do objeto ventilador. O deslocamento obedece um ângulo para impulsão definido por `Ângulo`, dando a impressão de serem ventilados, até saírem do palco. Este comportamento pode ser considerado uma especialização do comportamento `Impulsionar`, como ilustrado na figura 17.

O comportamento `Impulsionar` difere do `Ventilar` por deslocar apenas os objetos que forem tocados pelo objeto-impulsionador. `Ventilar` possui as características de `Impulsionar`, no entanto, a adição do campo-de-visão neste comportamento faz com que todos os objetos pertencentes à classe-alvo, que se encontram a uma distância do objeto-ventilador indicada por `Número1` sejam deslocados, caracterizando desta forma uma especialização do comportamento já existente `Impulsionar` gerando um novo comportamento.

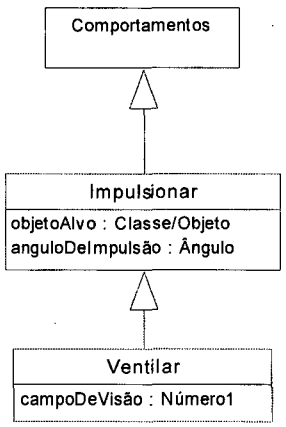

Figura 17: Classes envolvidas na criação do comportamento ventilar

Exemplo 2: Criação de Geradores Específicos

GerarPosiçãoAleatória (filho:Classe,
frequência:Número2,
exemplares:Número1)

GerarDireçãoAleatória (filho:Classe,
frequência:Número2,
exemplares:Número1)

Estes comportamentos consistem em gerar Número1 exemplares do objeto da classe filho a cada Número2 instantes de tempo. O objeto possuidor do comportamento GerarPosiçãoAleatória, deve gerar seus exemplares com sua direção e numa posição aleatória do palco (característica especializada). Diferentemente, um objeto com o comportamento GerarDireçãoAleatória, deve gerar seu exemplares na sua posição, estando estes com direção aleatória (característica epecializada).

GerarPosiçãoEspecífica (filho:Classe,
frequência:Número2,
exemplares:Número1,
posiçãoDeCriação:Ponto)

Este comportamento difere dos anteriores pelo fato dos exemplares serem gerados em uma posição específica do palco e com a mesma direção do objeto gerador (característica especializada). Esta posição é indicada por Ponto.

Estendendo-se a classe Gerar, pode-se obter os três diferentes comportamentos, mesmo estes desempenhando papéis diferenciados. O comportamento Gerar difere dos demais geradores, pelo fato dos exemplares serem gerados na posição e direção do gerador. A figura 18 ilustra a hierarquia de classes gerada através desta abordagem.

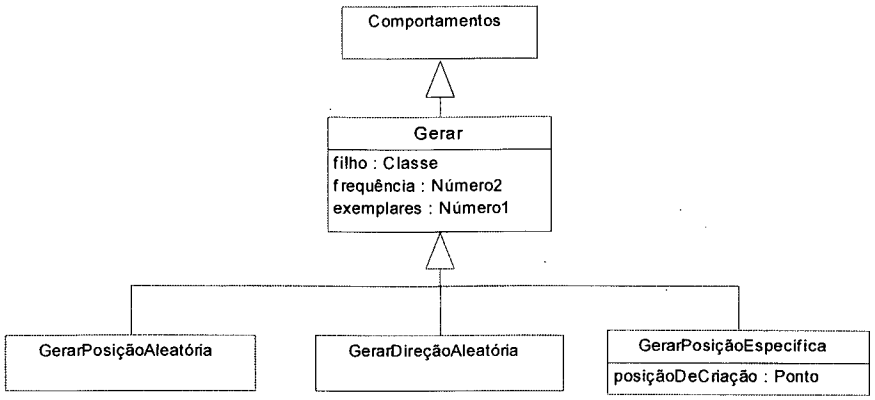

Figura 18: Hierarquia de classes da criação de geradores específicos

5.2.2 Composição ou Agregação

A relação de composição (agregação) permite que comportamentos sejam formados por agrupamento de comportamentos diferentes. Isto resulta na formação de uma nova classe (comportamento) como um agregado de classes preexistentes.

Desta forma, um *comportamento composto* consiste na agregação de dois ou mais comportamentos heterogêneos, inter-relacionados por meio de uma relação *parte_de* ou *componente_de* (Ricarte, 2001). Cada *comportamento componente* pode, por sua vez, ser um *comportamento composto*, o que resulta em uma hierarquia de composição. Esta hierarquia de composição (*parte_de*) se distingue semanticamente da hierarquia de especialização (*é_um)* (Rumbaugh, 1991).

Alguns comportamentos considerados na fase de análise dos projetos como complexos podem ser gerados através de composição ou agregação de comportamentos existentes. O comportamento a seguir é um exemplo dos que podem ser gerados utilizando esta abordagem e comportamentos *base.*

Exemplo: Criação do Comportamento Andar Em Ciclo

```
AndarEmCiclo(velocidadeDeDeslocamento:Número1
             tamanhoDoPasso:Número2,
             anguloParaMudançaDeDireçao:Ângulo)
```

Este comportamento consiste em um objeto que se move com velocidade `Número1`, mudando `Ângulo` na sua direção a cada `Número2` passos, resultando num comportamento em ciclo. Este comportamento é formado pela combinação dos comportamentos `Andar` e `Girar`, como ilustra a figura 19. `Andar` define que o objeto deve mover-se na mesma direção na velocidade indicada. `Girar` define que o objeto deve mudar sua direção de acordo com o ângulo especificado. Assim, `AndarEmCiclo` define quando o objeto que esta movendo-se deve mudar de direção.

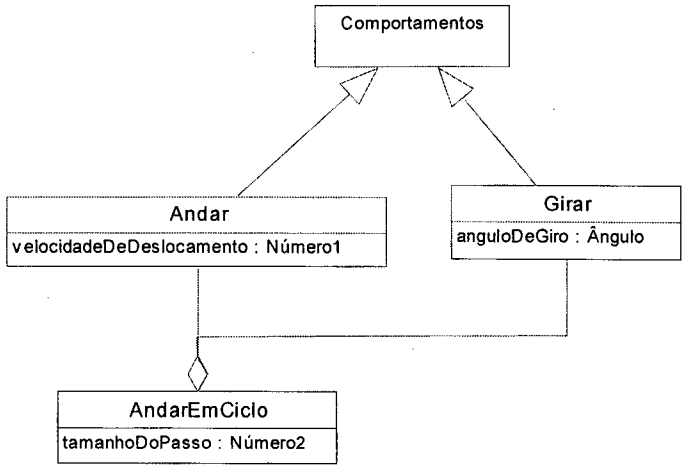

Figura 19: Classes envolvidas na criação do comportamento andarEmCiclo

5.2.3 Especialização de parâmetros

Como já mencionado, um novo comportamento herda suas propriedades de suas superclasses, entretanto, se a propriedade já se encontra definida em nível de subclasse, é esta a definição que prevalece. Esta propriedade denomina-se *overriding* (Palazzo, 2002).

Além da redefinição de um atributo, transmitido sob um nome particular, pode-se ter diferentes domínios, restrições ou *defaults*. O domínio poderia ser o mesmo ou mais restrito do que a especificação da superclasse. É possível manter o mesmo atributo substituindo algumas restrições ou *defaults* por outras especificações (Ricarte, 2001).

Quando um atributo ou método possui o mesmo nome na superclasse e na subclasse, e esta redefine a superclasse, tem-se um refinamento por especialização. De outra forma, o atributo ou método presente na subclasse aumenta o conjunto de atributos ou métodos herdados da superclasse. Assim, no nível de subclasse, tem-se refinamentos devidos a valor de atributo, valor *default*, restrição, domínio ou método (Palazzo, 2002).

O *overriding*, por sua vez, somente pode ser aplicado quando a propriedade especificada na classe herdeira possui o mesmo nome da propriedade herdada. Além destas, outras propriedades podem ser definidas na subclasse, aumentando o número de propriedades aplicáveis.

Exemplo: Especificação de Parâmetro no Comportamento AndarEmCiclo

AndarEmCicloVelocidade10 (`velocidadeDeDeslocamento:10,`
`tamanhoDoPasso:Número2,`
`anguloParaMudançaDeDireçao:Ângulo`)

Este comportamento consiste num objeto que se move com velocidade `10`, mudando `Ângulo` na sua direção a cada `Número2` passos, resultando num comportamento em ciclo. Assim, qualquer que seja o movimento resultante, o objeto sempre o realizará na mesma velocidade (10), caracterizando desta forma uma especialização de parâmetro da sua superclasse `AndarEmCiclo`, como ilustrado na figura 20. `AndarEmCiclo` e `AndarEmCicloVelocidade10` possuem as mesmas características e ações a serem desempenhadas, a única diferença é que o comportamento especializado deve mover-se com velocidade fixa para qualquer valores dos demais parâmetros.

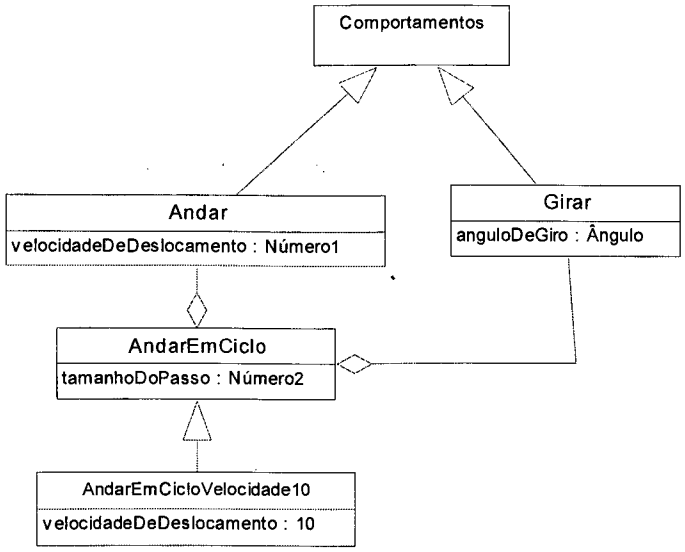

Figura 20: Hierarquia de classes do comportamento andarEmCicloVelocidade10

Com estas abordagens pode-se obter diferentes comportamentos e estes atribuídos a um objeto. No entanto, existe a possibilidade de interferências de comportamentos nas ações a serem desempenhadas pelos objetos. Estas interferências são resultantes da combinação de diferentes comportamentos ao mesmo objeto. A seção a seguir identifica as possíveis situações de interferência desta combinação de comportamentos.

5.3 Identificação de Interferências entre Comportamentos

Assumindo-se que o objeto pode ter vários comportamentos, pode-se identificar certas situações devidas à composição de alguns comportamentos. As combinações de comportamentos podem causar interferências nas ações a serem desempenhadas, podendo existir a hipótese da impossibilidade de co-ocorrência de alguns comportamentos.

Estas interferências podem ser classificadas analisando-se quando os comportamentos referenciam classes diferentes e quando referenciam a mesma classe. Para cada par de comportamentos que possuem uma classe como parâmetro, é

verificado se a classe é a mesma, prevendo-se o resultado da co-ocorrência destes comportamentos. A descrição das situações resultantes desta co-ocorrência são listados na tabela 4:

Tabela 4. – Situações resultantes quando na co-ocorrência de comportamentos

Situação	Descrição da Situação
1	***Comportamentos não interferentes***, comportamentos que se combinam e não interferem nas ações de outro.
2	***Comportamentos interferentes***, a soma dos dois comportamentos produz outro comportamento diferente dos anteriores.
3	***Comportamentos anuladores***, quando um comportamento anula outro.
4	***Comportamentos incompatíveis,*** os comportamentos se anulam mutuamente.

Este fato aponta para a necessidade de se analisar a lista de comportamentos atuais classificando as combinações de comportamentos de acordo com a tabela 4. Comprovou-se que podem ser atribuído diferentes comportamentos aos objetos, não existindo uma situação de impossibilidade de atribuição de determinado comportamento pela atribuição de outro. O que ocorre, no entanto, é a interferência de determinados comportamentos nas ações que outros deveriam desempenhar.

As situações resultantes da interferência ou atribuição de comportamentos ao mesmo objeto encontradas nesta fase de análise estão dispostas na tabela 5. Nesta tabela, um comportamento é comparado com outros observando-se a igualdade de classes ou não. Assim, obtêm-se as situações possíveis para cada par de combinação. Para esta investigação assume-se igualdade de valores nos parâmetros que ambos comportamentos possam ter, exceto quando o parâmetro for uma classe.

A simbologia adotada para representar as situações resultantes da co-ocorrência de comportamentos é a mesma descrita na tabela 4. Tem-se quatro possíveis

situações representadas por números de 1 à 4. A leitura de situações resultantes da combinação de pares de comportamentos contidos na tabela 5 adota a seguinte forma:

- o primeiro algarismo numérico (ex.:**1**2) refere-se a investigação considerando a existência de classes diferentes como parâmetro dos comportamentos.

- o segundo algarismo numérico (ex.:1**2**) refere-se a investigação considerando a mesma classe como parâmetro dos comportamentos.

Outro aspecto que deve ser observado é a ausência de uma classe como parâmetro em determinados comportamentos. Isto também é considerado na investigação de interferências de comportamentos e está representado na tabela 5 através do símbolo "*" (asterisco).

Tabela 5. – Situações resultantes da co-ocorrência de comportamentos.

	Absorver	Andar	Gerar	Girar	Impulsionar	Morrer	MudarImagem	Perseguir
Absorver	1*	14	1*	13	2	13	13	
Andar		1*	2*	1*	2*	1*	2*	
Gerar			1*	11	2*	11	13	
Girar				1*	3	1*	3*	
Impulsionar					12	11	22	
Morrer						11	33	
MudarImagem							11	

Com esta análise, pôde-se observar que, em algumas combinações os comportamentos possuem os mesmos parâmetros. Se considerarmos os valores dos parâmetros, além das classes, as situações anteriores podem se modificar, pois os parâmetros podem influenciar no desempenho das ações do comportamento. Deve-se, então, proceder com a mesma investigação considerando os diferentes valores para o mesmo parâmetro.

Como exemplo, pode-se citar um objeto que deve *perseguir* objetos de determinada classe e *fugir* destes mesmos objetos. Sabendo-se que os comportamentos *perseguir* e *fugir* possuem a seguinte sintaxe:

```
Perseguir(perseguirObjeto:Classe,
          velocidadeDeDeslocamento:Número1,
          campoDeVisão:Número2)

Fugir(fugirDe:Classe,
      velocidadeDeDeslocamento:Número1,
      campoDeVisão:Número2)
```

Se não forem considerados os valores dos parâmetros como `velocidadeDeDeslocamento`, ou se houver igualdade de valores para ambos, pode-se concluir que, os dois comportamentos se anulam. Por outro lado, se considerarmos valores diferentes para o parâmetro `velocidadeDeDeslocamento`, o comportamento, que possuir maior valor para este parâmetro prevalecerá. O resultado será a anulação do comportamento que possuir menor valor para o parâmetro.

Observa-se que a interferência de alguns comportamentos nas ações de outros causadas pelos valores de parâmetros, já ocorre com alguns dos comportamentos básicos. A tabela 6 indica as combinações de comportamentos em que os valores de parâmetros influenciam nas ações que devem ser desempenhadas pelo objeto.

Tabela 6. – Combinações de comportamentos em que os valores de parâmetros influenciam nas suas ações.

	Absorver	Andar	Gerar	Girar	Impulsionar	Morrer	MudarImagem	Perseguir
Absorver			**X**		**X**			**X**
Andar								
Gerar								**X**
Girar								
Impulsionar								
Morrer								**X**
MudarImagem								

Existem ainda, comportamentos que em que não se consegue definir claramente seus parâmetros genéricos. Na maioria dos casos, estes comportamentos são condicionados para sua ocorrência independentemente da sua definição. Esta questão será tratada na seção a seguir.

5.4 Comportamentos Condicionados

Alguns dos comportamentos identificados, não podem ser trabalhados segundo as abordagens apresentadas anteriormente. Isto se deve ao fato de que estes comportamentos são condicionados para a sua execução e/ou não possuem parâmetros claros que satisfaçam uma situação genérica, por isso, não se consegue definir claramente parâmetros genéricos para eles.

Um destes comportamentos é o `Morrer`, cuja ocorrência depende de diferentes condições, como exemplo: quando é tocado, toca algo, encontra-se nos limites do palco ou determinado ponto do mesmo, esgota seu tempo de vida, dentre outras. Para cada uma destas condições, o comportamento tem uma definição distinta.

Uma forma de modelar este comportamento, é enumerar as possíveis condições em que o objeto pode morrer, definido-as como parâmetros opcionais do comportamento `Morrer`. De tal forma, que sendo satisfeita uma ou mais condições, o objeto desempenharia suas ações de acordo com o comportamento e suas condições.

Por outro lado, poder-se-ia considerar que o objeto morre em determinadas situações e em uma situação específica as propriedades do comportamento que diferenciam do original, são redefinidas. Isto se deve ao fato de que as propriedades definidas em nível mais baixo se sobrepõem ou refinam as mesmas propriedades herdadas da superclasse. Assim, as condições para o objeto morrer seriam implementadas à medida que for surgindo a necessidade da sua criação. Estas então são incorporadas à biblioteca, juntamente com os demais comportamentos.

Estas abordagens para modelar estes tipos de comportamentos podem ser utilizadas neste trabalho, não sendo consideradas ideais para o mesmo. Pois, para

atender à proposta em que estão inseridos os demais comportamentos, é necessário que estes contemplem uma situação genérica, podendo ser especializados se aplicados em uma situação específica.

Outra alternativa é considerar que um objeto não pode se auto-destruir. Assim o comportamento `Morrer` seria substituído pelo `Destruir`. Um objeto então morreria quando outro o destruísse. Esta abordagem, assim como a anterior, não atende a maioria das possibilidades do comportamento `Morrer`, apesar de que, nesta alternativa se teria situações mais genéricas que poderiam ser especializadas. Entretanto, não se soluciona a concepção de comportamento condicionado.

O que ocorre realmente é que os comportamentos condicionados possuem como parâmetro de ocorrência o *"quando"* e o valor deste parâmetro é um trecho de código de uma linguagem de programação, descritos da seguinte forma: **Morre**(`quando:Código`). Este código satisfaria a condição atual para a ocorrência do comportamento no caso específico.

Pode-se generalizar que os comportamentos que estão sendo considerados neste trabalho podem também ter o parâmetro *quando* como opcional. Esta abordagem resolveria várias situações de difícil modelagem.

Estes fatores fazem surgir um novo enfoque para tratamento dos comportamentos. Pode-se assumir que os comportamentos gerados ou disponibilizados através da biblioteca, servem como elementos de apoio ou complemento à linguagem atualmente disponível no Mundo dos Atores. A definição dos comportamentos no método *define papel* (Mariani, 1998) continuaria, e não seriam descartados o uso das sentenças existentes na linguagem para a elaboração das condições de ocorrência do comportamento em situações específicas.

No entanto, a proposta inicial deste trabalho é compor os comportamentos a partir de uma interface gráfica e não na forma textual como está sendo feito aqui. Com a existência de comportamentos condicionados não é possível que a atribuição de comportamentos a objetos seja feita somente através de uma interface gráfica. Da maneira em que se encontram atualmente modelados os comportamentos, mesmo com a

atribuição a objetos na forma textual apresentam-se mais satisfatórios em relação aos objetivos traçados, considerando-se a programação existente atualmente na ferramenta Mundo dos Atores.

Os conceitos abordados até o momento, são importantes para a construção de aplicações usando comportamentos integrantes da biblioteca proposta. A seção a seguir, descreve alguns exemplos de aplicações já implementadas na ferramenta Mundo dos Atores, e propostas de implementação usando os comportamentos da biblioteca e demais conceitos trabalhados.

5.5 Exemplos de Aplicações usando Comportamentos da Biblioteca

As aplicações usadas para exemplificar a utilização dos comportamentos propostos para integrar a biblioteca de padrões de comportamentos são semelhantes aos exemplos apresentados no capítulo 3, e a forma de atribuição usada é textual. Isto se deve ao fato da possibilidade de comparações entre a forma de implementação atual dos comportamentos na ferramenta Mundo dos Atores e a proposta de uso dos comportamentos da biblioteca. Estes exemplos são apresentados nas seções a seguir.

5.5.1 Projeto Missão no Espaço

Missão no Espaço é um jogo muito semelhante ao projeto Invasores do Espaço descrito no capítulo 3 (item 3.1.1.1), onde se parte da idéia da existência de uma nave-mãe que lança naves menores. Cada nave menor lança bombas que caem para o chão, onde um canhão controlado pelo usuário está localizado.

Os objetos existentes neste jogo são: Canhão, Canhão Explodido, Nave Mãe, Nave Mãe Explodida, Míssil, Míssil Explodido, Bomba, Bomba Explodida, Nave e Nave Explodida. As classes representando as propriedades destes possuem os mesmos nomes.

O `Canhão` é movido para a esquerda e para a direita pelo usuário, não podendo

encontrar-se fora do palco. Assim, quando ele está fora do palco, ele anda para trás de volta ao palco. Sua função é disparar mísseis quando indicado pelo usuário. Quando for tocado por uma `BombaExplodida` ou `NaveExplodida`, ele se torna um `CanhãoExplodido`. Sendo que um `CanhãoExplodido` deve apenas existir por uma breve fração de tempo após sua criação e então desaparecer. As descrições dos comportamentos e seus parâmetros estão no ANEXO2.

```
definePapel"na classe Canhão"
baterVoltarEParar(2,180,35)
tornarSe(BombaExplodida, CanhãoExplodido)
tornarSe(NaveExplodida, CanhãoExplodido)

definePapel"na classe CanhãoExplodido"
morrerEm(10)
```

A `Nave Mãe` movimenta-se aleatoriamente pelo palco. Em determinados tempos lança uma nave menor. Ao tocar um `MíssilExplodido`, ela se torna uma `NaveMãeExplodida`, existindo por algumas frações de tempo e então desaparecendo:

```
definePapel"na classe NaveMãe"
andarAleatoriamente(3)
gerarDireçãoAleatória(Nave, 15, 1)
tornarSe(MíssilExplodido, NaveMaeExplodida)

definePapel"na classe NaveMãeExplodida"
morrerEm(10)
```

Uma `Nave` deve mover-se pelo palco, soltar bombas, sentir se foi tocada pelo chão ou por um `MíssilExplodido`. Torna-se uma `NaveExplodida`, se tocada por um `MíssilExplodido`. Sendo que, uma `NaveExplodida` deve existir por alguns instantes de tempo após sua criação e então desaparecer:

```
definePapel"na classe Nave"
andar(4)
gerarDireçãoEspecífica(Bomba, 0.2, 1)
tornarSe(MissilExplodido, NaveExplodida)
tornarSe(limitesDoPalco, NaveExplodida)

definePapel"na classe NaveExplodida"
morrerEm(10)
```

A `Bomba` e o `Míssil` devem mover-se sempre na mesma direção, até saírem dos

limites do palco. Ao tocar Nave e Míssil a Bomba torna-se uma BombaExplodida. Da mesma forma, um Míssil torna-se MíssilExplodido ao tocar o Canhão. Sendo que o Míssil é mais veloz que a Bomba:

```
definePapel "na classe Bomba"
andar(3)
tornarSe(Nave, BombaExplodida)
tornarSe(Míssil, BombaExplodida)

definePapel "na classe Míssil"
andar(4)
tornarSe(Canhão, MíssilExplodido)
```

Uma BombaExplodida, como um MíssilExplodido, deve existir por uma fração de tempo após sua criação e então desaparecer:

```
definePapel"na classe MíssilExplodido"
morrerEm(10)

definePapel"na classe BombaExplodido"
morrerEm(10)
```

A hierarquia de classes envolvidas neste exemplo está ilustrada na figura 21. Foram criados novos comportamentos segundo as abordagens descritas no item 5.2, e aplicadas algumas das formas propostas para o tratamento de comportamentos condicionados discutidas no item anterior. Os comportamentos foram gerados a partir dos comportamentos definidos como básicos por este trabalho.

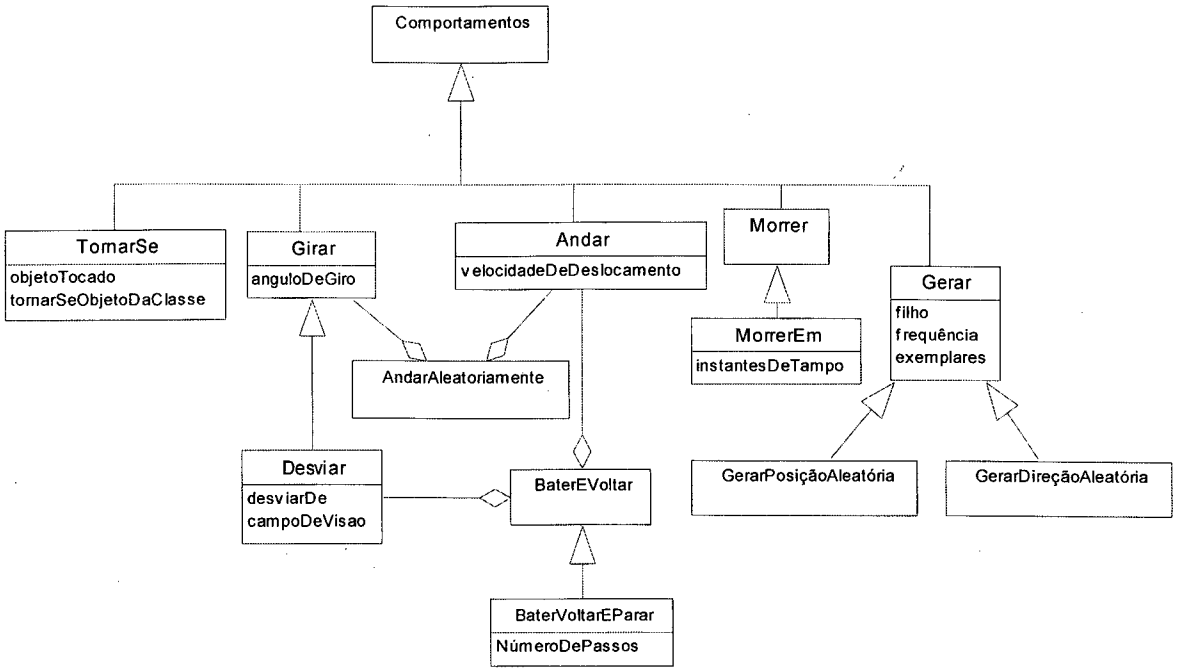

Figura 21: Hierarquia de classes envolvidas no Projeto Missão no Espaço

5.5.2 Projeto Presas e Predadores

Presas e Predadores, é um jogo semelhante ao projeto Meio Ambiente do capítulo 3 (item 3.1.1.2), onde existe um ambiente habitado por presas e predadores, onde cada um possui uma função diferenciada dos demais. Os elementos constituintes deste ambiente são: predador, presa adulta, presa jovem e grama.

A `Grama` deve reproduzir-se a cada 200 unidades de tempo, gerando um novo exemplar. E morrer quando seu tempo de vida atingir 750 unidades de tempo:

```
definePapel "na classe Grama"
gerarAleatoriamente(Grama, 200,1)
morrerEm(750)
```

A `PresaJovem` deve andar aleatoriamente pelo palco e quando encontrar uma `PresaAdulta` nas proximidades deve seguí-la. Ao encontrar o `Predador`, deve fugir deste. Ainda, deve alimentar-se de `Grama` quando encontrá-la. Após 1000 unidades de tempo de sua criação, transforma-se em `uma PresaAdulta`:

```
definePapel "na classe PresaJovem"
andarAleatoriamente(10)
perseguir(PresaAdulta, 12, 200)
fugir(Predador, 12, 200)
absorver(Grama)
tornerSeEm(1000,PresaAdulta)
```

A `PresaAdulta` deve andar aleatoriamente pelo ambiente, gerar `PresasJovens` a cada 300 unidades de tempo e fugir do predador quando este estiver a uma distância menor ou igual a 400. Morre em 900 unidades de tempo após ser criada:

```
definePapel "na classe PresaAdulta"
andarAleatoriamente(13)
gerarDireçãoAleatória(PresaJovem, 300, 2)
fugir(Predador, 17, 400)
MorrerEm(900)
```

O `Predador` anda aleatoriamente pelo palco, até encontrar uma `PresaJovem` ou `Adulta` nas proximidades. Encontrando-as, deve persegui-las e absorve-las, dando prioridade a `PresaAdulta`:

```
definePapel "na classe Predador"
andaAleatoriamente(15)
perseguir(PresaAdulta, 17, 400)
perseguir(PresaJovem, 17, 200)
absorver(PresaAdulta)
absorver(PresaJovem)
```

Os comportamentos envolvidos na solução de implementação deste exemplo estão na figura 22. Como no exemplo anterior, foram aplicadas as abordagens discutidas neste capítulo.

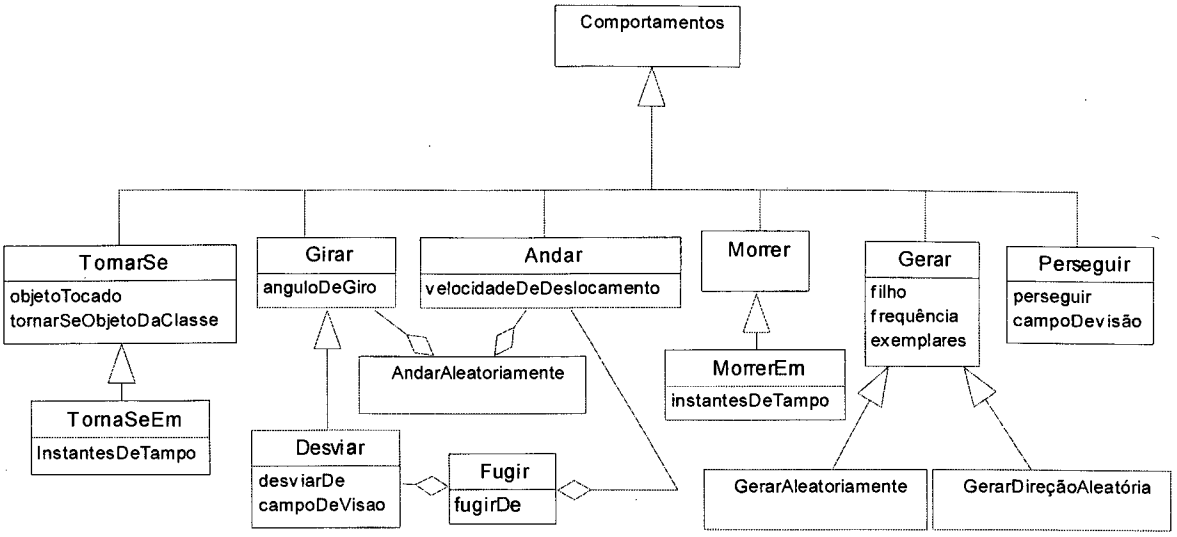

Figura 22: Hierarquia de classes envolvidas no Projeto Presas e Predadores

Com a utilização destes comportamentos, tem-se um ganho em relação a linhas de códigos e entendimento da lógica envolvida no desenvolvimento de aplicações. O desenvolvedor atém-se ao tema em estudo e os conceitos envolvidos na aplicação a ser desenvolvida. Grande parte dos comportamentos podem ser criados através das abordagens discutidas nos itens anteriores, e alguns já se encontram modelados, estando isenta a elaboração da lógica de funcionamento dos mesmos.

Assim, o desenvolvedor sem conhecimento específico na linguagem, escolhe comportamentos apropriados aos objetos de sua aplicação e fornece os valores necessários à ocorrência dos mesmos.

O tempo desprendido anteriormente na construção da lógica de funcionamento do comportamento pode ser aplicado no tema em estudo. Isto pode ainda motivar o desenvolvedor para que não abandone a criação de aplicações e de seus próprios comportamentos.

6 Conclusões e Sugestões Para Trabalhos Futuros

A maioria dos projetos (aplicações) desenvolvidos pelos alunos e analisados neste trabalho envolve algum tipo de jogo. Observa-se que, os diferentes projetos de determinado grupo de trabalho, apresentam comportamentos de objetos muito semelhantes. Após estes comportamentos serem trabalhados no sentido de se obter uma hierarquia de comportamentos, observa-se a existência de um número pequeno de comportamentos. Existe sim, uma variação de determinados aspectos dos comportamentos, ou comportamentos que podem ser gerados através de combinação de outros.

Com a observação das ações executadas pelos objetos no palco (critério de investigação II), obtém-se comportamentos expressivos, que podem ser aplicados em diferentes contextos. Esta conclusão obtém-se comparando-se com o critério de análise onde verifica-se as linhas de código correspondentes aos comportamentos relacionando-as com os comportamentos dos objetos observados em tempo de execução (critério de investigação I). Porém, o critério de investigação I não deve ser descartado por possibilitar a identificação de comportamentos que podem não ser visualizados em determinado momento da execução. Fica evidente que a ênfase deve estar no critério de investigação II, combinando-se aspectos encontrados no critério de investigação I.

Podem ser atribuídos vários comportamentos ao mesmo objeto. Não existem comportamentos que não podem ser atribuídos em restrição a outros. O que ocorre é que, algumas combinações de comportamentos podem gerar

interferências nas ações a serem desempenhadas. E, isto vai depender dos valores de parâmetros dos comportamentos.

A maioria dos comportamentos definidos neste trabalho como sendo básicos, podem ser aplicados a uma situação genérica fornecendo apenas os valores necessários para seus parâmetros. Estes podem ser especializados para uma situação específica estendendo-se a classe, agregando ou compondo características e especializando parâmetros. Com isto, pode-se gerar um número expressivo de comportamentos.

No entanto, existem comportamentos que são condicionados para sua ocorrência. Ou seja, para que o objeto desempenhe as ações do comportamento, é necessário que seja satisfeita a condição definida para execução do mesmo. Normalmente, existem inúmeras situações para a ocorrência deste tipo de comportamento, não sendo encontrada uma definição genérica. Conclui-se neste trabalho que o parâmetro para estes comportamentos é uma situação definida por *quando*, sendo esta uma situação complexa (expressa por código de programação) que satisfaz a condição para ocorrência do mesmo.

A existência destes comportamentos condicionados é responsável pela mudança de enfoque na proposta da biblioteca. Assume-se que os comportamentos gerados podem ser complementos à linguagem de programação utilizada, e não uma substituição de parte desta.

Os usuários da ferramenta continuariam condicionados a linhas de códigos para a construção de aplicações. No entanto, haveria uma redução considerável na programação envolvida baseando-se nos projetos já implementados. Isto se deve ao fato de que o desenvolvedor deverá programar apenas situações específicas envolvendo comportamentos condicionados e aquelas que não foram previstas na criação da biblioteca. Mesmo assim, consegue-se reduzir o tempo de dedicação no processo de criação de aplicações, onde o desenvolvedor preocupa-se mais com os conceitos que devem ser apresentados e não a forma de representar utilizada pela ferramenta. Da mesma forma, a carga cognitiva envolvida deixa de ser um fator preocupante, que tem influência direta no processo de criação, pois, seria

desnecessária a criação de todos os comportamentos necessários na construção de aplicações.

Este trabalho aponta para a tendência de fornecer metáforas acessíveis ao entendimento de programação de aplicações para não-especialistas no assunto. O desafio desta tendência consiste em pegar numa boa concepção de linguagem e encontrar um conjunto coerente de concretizações, que abarquem os elementos da linguagem a serem utilizados. Em outras palavras, disponibilizar abstrações de alto nível para expressão de programas, e disponibilizar sistematicamente elementos análogos a estas abstrações, mas que sejam concretos, intuitivos e fáceis de aprender, não é tarefa fácil. Estes aspectos estão motivando várias pesquisas objetivando fornecer metáforas acessíveis ao entendimento de crianças, para servirem de apoio às atividades de aprendizado nas escolas.

Com esta tendência de programação visual aliada com conceitos de ensino não-sequencial e dirigido por projetos, quando trabalhada com estudantes objetivando a aprendizagem de conceitos apresentados, direciona a aprendizagem para o modelo construtivista, sendo esta a busca de várias escolas.

6.1 Sugestões para Trabalhos Futuros

Como propostas para trabalhos futuros sugere-se:

a) Estudo aprofundado dos comportamentos definidos como condicionados. Com isto, pode-se analisar as soluções para adaptar estes comportamentos à proposta em que estão inseridos os demais. Aplicar estas soluções à biblioteca e selecionar a melhor alternativa.

b) Considerar aspectos dinâmicos (como níveis de energia ou vidas) na concepção de comportamentos. Identificar as maneiras de se tratar esta abordagem, incorporando este aspecto ao conjunto de comportamentos existentes na biblioteca.

c) Documentar os padrões de comportamentos num formato simplificado para se escrever um *design pattern*. Não há necessidade de todos os elementos citados no capítulo 2 (ítem 2.5) na sua descrição. No entanto, alguns componentes devem ser facilmente reconhecidos como nome e classificação, propósito, motivação, aplicabilidade, participantes, colaborações, diagrama, conseqüências, implementação, exemplos e padrões de projeto afins.

d) Proposta de criação de uma nova linguagem que contemple os comportamentos condicionados, facilitando a criação de aplicações de maneira que o usuário não necessite utilizar os comandos e a forma de programar existentes atualmente na ferramenta Mundo dos Atores.

7 Referências Bibliográficas

Alexander, Christopher (1993). An Introduction for Object-Oriented Designers. SUNY Oswego / NY CASE Center Last Content Change: 11 December. http://g.oswego.edu/dl/index.html, último acesso 20/09/2001.

Alexander, Christopher et al (1977). A Pattern Language. Oxford University.

Ambler, Scott W. (1998). Process Patterns: Building Large-Scale Systems Using Object Technology. Cambridge University Press/SIGS Books, July.

Ambler, Scott W. (1999). More Process Patterns: Delivering Large-Scale Systems Using Object Technology. Cambridge University Press/SIGS Books.

Booch, G. (1991). Object-Oriented Design with Applications. Benjamin Cummings, Redwood.

Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Wiley,M. Stal John (1996). Pattern-Oriented Software Architecture : A System of Patterns. John Wiley & Sons, Inc. UK.

Coplien, James O & Schmidt, Douglas (1995). Pattern Languages of Program Design. Addison-Wesley.

Coplien, James O. (1994). Software Design Patterns: Common Questions and Answer. Available from Patterns Home Page at <http://st.www.cs.uiuc.edu /users/patterns/patterns.html>, último acesso 24/10/2001.

Coplien, James O.; Vlissides, John M. and Kerth, Norman L. (1996). Pattern Languages of Program Design 2. Addison-Wesley.

Erickson, Hans-Erik & Penker, Magnus (1998). UML Toolkit. New York. John Wiley & Sons, Inc. 397 p.

Fairley, R. (1986). Software Engineering Concepts. McGraw-Hill.

Furlan, José Davi (1995). Reengenharia da Informação: do Mito a Realidade. São Paulo, Makron Books.

Gamma, Erich; Helm, Richard; Johnson, Ralph and Vlissides, John (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Kahn, Ken (1995). Metaphor Design: Case Study of an Animated Programming Environment. Incluída em Proceedings of the Computer Game Developers Conference.

Larman, Craig (2000). Utilizando UML e Padrões: Uma introdução à análise e ao projeto Orientados a Objetos. Tradução Luiz A. Meirelles Salgado. Porto Alegre. Bookman. 492 p.

Mariani, A. Carlos (1998). O Mundo dos Atores: uma perspectiva de introdução à programação orientada a objetos. Anais do Simpósio Brasileiro de Informática na Educação (SBIE). Fortaleza, Ceará.

Mariani, A. Carlos (2001). O Mundo dos Atores. http://www.inf.ufsc.br/poo/atores, último acesso 16/03/2002.

Nova Escola (1995). O tira-teima do construtivismo - 50 grandes e pequenas dúvidas esclarecidas. Março,pp. 8-13.

Palazzo, Luiz A. M. (2002). Modelagem Conceitual Orientada a Objetos. Universidade Católica de Pelotas – Biblioteca Virtual, Escola de Informática. Disponível em http://esin.ucpel.tche.br/bbvirt/art/ObjRevis.htm, último acesso 15/03/2002.

Papert, S. (1980). Mindstorms: Children, Computers and Powerful Ideas. New York. Basic Books.

Piaget, Jean (1967). A psicologia da inteligência. Lisboa. Editora Fundo de Cultura SA.

Pressman, Roger S. (1995). Engenharia de Software. Makron Books.

Ricarte, Ivan Luiz Marques (2001). Programação Orientada a Objetos com C ++ - Anotações de Aulas. Departamento de Engenharia de Computação e Automação Industrial, Universidade Estadual de Campinas. Disponível em http://www.dca.fee.unicamp.br/~ricarte/POOCPP/POOCPP.html, último acesso 16/03/2002.

Rumbaugh, James; Blaha, Michael; Premerlani, William; Eddy, Frederick; Lorenson William (1991). Object-Oriented Modeling and Design. 1ª edição, *Prentice Hall*. 500 p.

Silva, R. P. (2000). Suporte ao Desenvolvimento e uso de frameworks e componentes. Tese de Doutorado, PPGC - UFRGS, Porto Alegre.

Silva, Thiago Henrique (2001). Criação de uma Biblioteca de Padrões de Comportamento de Atores para o Desenvolvimento de Aplicações Educacionais. Relatório de Bolsa de Iniciação Científica.

Wazlawick, R. S. & Mariani, A. C. (2000). Actors and stage metaphor for introducing object-oriented programming. In D. Benzie & D. Passey (Eds.), *Proceedings of the IFIP TC3 Open International Conference on Educational Uses of Information and Communication Technologies – IFIP World Computer Congress*, 247-252.

Wazlawick, R. S.; Rosatelli, M. C.; Ramos, E. M. F.; Cybis, W. A.; Storb, B. H.; Schumacher, V. R. N.; Mariani, A. C.; Kirner, T.; Kirner, C.; Fagundes, L. C. (2001). Providing More Interactivity to Virtual Museums: A Proposal for a VR Authoring Tool. *Presence Teleoperators And Virtual Environments*, MIT Press, Cambridge, USA. 10(6):647-656. December.

Wazlawick, R. S.; Kirner, T.; Fagundes, L. C.; Rosatelli M. C.; et al. (2001). museuVirtual: An authoring tool for the creation of museums in virtual reality to support collaborative learning through the Internet. Universidade Federal de Santa Catarina, Centro de Pesquisas de São Carlos e Universidade Federal do Rio Grande do Sul. Relatório Final do Projeto.

Yamada, S.; Hong, J. & Sugita, S. (1995). Development and evaluation of hypermedia for museum education: Validation of metrics. *ACM Transactions on Computer-Human Interaction*, **2**(4), 284-307.

ANEXO 1 – Lista de alguns padrões de comportamentos encontrados nos projetos

Nome do Comportamento	Descrição do Comportamento
Absorver	o objeto absorve outro ao tocá-lo. O objeto absorvido desaparece.
AndarAleatoriamente	o objeto movimenta-se pelo palco, com uma velocidade estabelecida, mudando sua direção aleatoriamente numa dada freqüência.
AndarAleatoriamenteBaterEVoltar	o objeto move-se pelo palco, com uma velocidade estabelecida, mudando sua direção aleatoriamente numa dada freqüência, até encontrar um determinado obstáculo. Ao encontrá-lo, movimenta-se na direção contrária, continuando o movimento inicial.
AndarAtéObstáculo	objeto move-se numa dada direção com velocidade especificada até encontrar um determinado obstáculo.
AndarEAparecerPontoInicial	o objeto parte de um ponto, move-se na velocidade especificada e, sempre que o mesmo passar por determinado ponto do palco, desaparece e aparece na sua posição inicial.
AndarEmCiclo	o objeto movimenta-se com velocidade fornecida até uma dada distância, mudando sua direção com o ângulo especificado, repetindo-se o movimento inicial.
AndarEMorrer	o objeto movimenta-se numa dada direção e velocidade, desaparecendo ao tocar os limites do palco.
AndarEParar	o objeto move-se numa dada direção e velocidade, parando em um determinado ponto.
AndarERetornarAoCentroDoPalco	o objeto movimenta-se numa dada direção e velocidade. Sempre que o objeto estiver em um determinado ponto, movimenta-se até o centro do palco.

Nome do Comportamento	Descrição do Comportamento
BaterEMorrer	o objeto movimenta-se numa dada direção e velocidade, desaparecendo ao tocar determinado objeto ou os limites do palco.
BaterEMovimentar	o objeto movimenta-se numa dada direção e velocidade. Ao tocar um determinado objeto, impulsiona-o numa direção, mantendo seu movimento inicial.
BaterMovimentarEVoltar	o objeto movimenta-se numa dada direção e velocidade. Ao tocar um determinado objeto, impulsiona-o numa direção, mudando sua própria direção.
BaterEMudarImagem	o objeto movimenta-se, numa dada direção e velocidade, e altera a imagem de determinado objeto ao tocá-lo.
BaterEVoltar	o objeto movimenta-se, numa dada direção e velocidade. Ao tocar determinado objeto ou os limites do palco, altera a sua direção.
BaterLateraisDoPalcoEVoltar	o objeto movimenta-se, numa dada direção e velocidade. Ao tocar as laterais do palco, altera a sua direção. Desaparecendo caso toque os limites superior e inferior do palco.
BaterEVoltarPosiçaoInicial	o objeto movimenta-se, numa dada direção e velocidade. Ao tocar determinado objeto ou estiver em um determinado ponto, retorna ao ponto inicial.
BaterMorrerEDestruir	o objeto movimenta-se, numa dada direção e velocidade. Ao tocar determinado objeto, ambos desaparecem.
BaterMovimentarEMorrer	o objeto movimenta-se, numa dada direção e velocidade. Ao tocar determinado objeto, movimenta-o e desaparece.
BaterMudarImagemEVoltar	o objeto movimenta-se, numa dada direção e velocidade. Ao tocar determinado objeto ou os limites do palco, altera sua imagem e direção.
Desviar	o objeto movimenta-se, numa dada direção e velocidade. Ao aproximar-se de determinado objeto, altera sua direção.
Fugir	o objeto movimenta-se, numa dada direção e velocidade. Ao aproximar-se de determinado objeto, altera sua direção afastando-se do mesmo.
Gerador	o objeto gera exemplares de um objeto em dada freqüência. Os exemplares são gerados na posição e direção do gerador.

Nome do Comportamento	Descrição do Comportamento
GeradorDireçãoAleatória	o objeto gera exemplares de outro objeto em dada freqüência. Os exemplares são gerados na posição do gerador e com direção aleatória.
GeradorPosiçãoAleatória	o objeto gera exemplares de outro objeto em dada freqüência. Os exemplares são gerados com direção do gerador e com posição aleatória.
GeradorPosiçãoEspecífica	o objeto gera exemplares de outro objeto em dada freqüência. Os exemplares são gerados numa posição específica e com a direção do gerador.
Morrer	o objeto desaparece numa dada condição.
Perseguir	o objeto movimenta-se, numa dada direção e velocidade. Ao encontrar determinado objeto, altera sua direção aproximando-se do mesmo.
Ventilar	ao encontrar objetos que estejam nas proximidades, estes são deslocados dando a impressão de serem ventilados, até saírem do palco.
Impulsionar	ao tocar um objeto, este é empurrado em direção contrária.

ANEXO 2 – Lista de alguns comportamentos com os parâmetros sugeridos.

Nome	Parâmetros	Descrição
Absorver	absorveObjeto:Classe	o objeto ao tocar qualquer outro objeto indicado pela `Classe`, absorve-o, sendo que o absorvido desaparece.
Andar	velocidadeDeDeslocamento:Número1	o objeto se movimenta com velocidade indicada por `Número1`, sempre na mesma direção até que encontre os limites do palco.
AndarAleatoriamente	velocidadeDeDeslocamento:Número1	o objeto se movimenta com velocidade `Número1`, mudando sua direção aleatoriamente, até que se esgote seu tempo de vida.
AndarEmCiclo	velocidadeDeDeslocamento:Número1 tamanhoDoPasso:Número2 anguloParaMudançaDeDireçao:Ângulo	consiste em um objeto que anda com velocidade `Número1`, mudando sua direção com o ângulo especificado a cada `Número2` passos.
BaterEVoltar	velocidadeDeDeslocamento:Número1 anguloParaMudançaDeDireçao:Ângulo	o objeto movimenta-se com velocidade `Número1`, e sempre que bater em um dos limites do palco, este muda `Ângulo` na sua direção e retorna seu deslocamento.
BaterVoltarEParar	velocidadeDeDeslocamento:Número1 anguloParaMudançaDeDireçao:Ângulo númeroDePassos:Número2	o objeto movimenta-se com velocidade `Número1`, ao bater em um dos limites do palco, muda `Ângulo` na sua direção e volta `Número2` passos.

Nome	Parâmetros	Descrição
Destruir	objetoAlvoParaDestruição:Classe	ao encontrar um objeto `Classe`, `este` é destruído, ou seja, encerra sua execução no palco desaparecendo.
Desviar	desviarDe:Classe campoDevisão:Número1 anguloParaMudançaDeDireção:Ângulo	ao encontrar um objeto `Classe`, que esteja a uma distância menor ou igual a `Número1`, o objeto muda `Ângulo` na sua direção.
Fugir	fogeDe:Classe velocidadeDeDeslocamento:Número1 campoDevisão:Número2	o objeto deve fugir dos objetos `Classe`, com velocidade `Número1`, sempre que estiverem a uma distância igual ou inferior a `Número2`.
Gerar	filho:Classe frequência:Número2 exemplares:Número1	consiste em gerar `Número1` exemplares do objeto `Classe` a cada `Número1` instantes de tempo. Os exemplares são gerados na posição e direção do gerador.
GerarAleatoriamente	filho:Classe frequência:Número2 exemplares:Número1	consiste em gerar `Número1` exemplares do objeto `Classe` a cada `Número2` instantes de tempo, em uma posição e direção aleatórias.
GerarDireçãoAleatória	filho:Classe frequência:Número2 exemplares:Número1	o objeto gera `Número1` exemplares do objeto `Classe` a cada `Número2` instantes de tempo. Os exemplares são gerados na posição do gerador e com direção aleatória.
GerarDireçãoEspecífica	filho:Classe frequência:Número2 exemplares:Número1 direçãoDeCriação:Ângulo	o objeto gera `Número1` exemplares do objeto `Classe` a cada `Número2` instantes de tempo. Os exemplares são gerados na posição do gerador e com direção `Ângulo`.
Girar	`anguloDeGiro:Ângulo`	o objeto muda sua direção obedecendo `Ângulo`.

Nome	Parâmetros	Descrição
Impulsionar	objetoAlvo:Classe anguloDeImpulsão:Ângulo	ao encontrar qualquer objeto `Classe`, estes são deslocados obedecendo `Ângulo` dando a impressão de serem empurrados
Mola	objetoRepulsivo:Classe anguloDeMudançaDeDireção:Ângulo velocidadeDeDeslocamento:Número1	o objeto se movimenta na mesma direção a uma velocidade `Número1` até encontrar um objeto `Classe`, mudando sua direção de deslocamento para `Ângulo`.
Morrer		o objeto desaparece.
MorrerEm	InstantesDeTempo:Número1	O objeto desaparece em `Número1` instantes de tempo após ser criado.
MudarImagem	objetoMutante:Classe novaImagem:bitmap	o objeto `Classe`, troca a sua imagem atual para `bitmap`..
Perseguir	perseguir:Classe velocidadeDeDeslocamento:Número1 campoDevisão:Número2	consiste em perseguir o objeto `Classe` que está a uma distância igual ou menor que `Número2` com velocidade `Número1`.
TornarSe	objetoTocado:Classe tornarSeObjetoDaClasse:Classe1	ao tocar objeto `Classe`, este transforma-se em um `Classe1`, assumindo as características desta nova classe.
TornarSeEm	InstantesDeTampo:Número1 tornarSeObjetoDaClasse:Classe	em `Número1` `instantes de tempo` após ser criado, o objeto torna-se um objeto `Classe`.
Ventilador	objetoAlvo:Classe anguloDeImpulsão:Ângulo campoDeVisão: Número1	ao encontrar objetos `Classe`, que estejam a uma distância menor ou igual a `Número1`, estes são deslocados obedecendo a `Ângulo` graus, dando a impressão de serem ventilados, até saírem do palco.